Dharmsinh Desai University

Department of MCA

Semester - II

Data Structures

Termwork Report
Submitted by

Roll No	Name
MCA034	Aditya .S. Nair

1 Write a Program to Merge a linked list into another linked list at alternate positions [Given two linked lists, insert nodes of second list into first list at alternate positions of first list. For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of the second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.]

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
} *new, *head1, *head2, *tail1, *tail2;
typedef struct node node;
void create_node()
{
   new = (node *)malloc(1 * sizeof(node));
   if (new == NULL)
   {
      printf("Memory Allocation Falied.");
      return;
   }
   else
   {
      new->next = NULL;
      printf("Enter data for the node : ");
      scanf("%d", &new->data);
   }
}
void insert_last(node **head, node **tail)
{
   create_node();
   if (*tail == NULL)
   {
      *head = new;
      *tail = new;
   }
   else
   {
      (*tail)->next = new;
      *tail = new;
      new = NULL;
   }
}
void display(node *head)
{
   printf("\nHead -> ");
   node *temp = head;
```

```
    while (temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("Tail\n");
}
void merge(node **h1, node **h2)
{
    node *temp_t1 = *h1; // Traversal for list 1
    node *temp_t2 = *h2; // Traversal for list 2
    node *temp_1, *temp_2; // Temporary nodes for swapping

    while (temp_t1 != NULL && temp_t2 != NULL)
    {
        temp_1 = temp_t1->next;
        temp_2 = temp_t2->next;

        temp_t1->next = temp_t2;
        temp_t2->next = temp_1;

        temp_t1 = temp_1; // Move to the next node in list 1
        temp_t2 = temp_2; // Move to the next node in list 2
    }
}
int main()
{
    int no_l1, no_l2;
    printf("\nEnter number of nodes in list 1 : ");
    scanf("%d", &no_l1);
    for (int i = 0; i < no_l1; i++)
    {
        printf("\nElement : %d ", i + 1);
        insert_last(&head1, &tail1);
    }

    printf("\nEnter number of nodes in list 2 : ");
    scanf("%d", &no_l2);
    for (int i = 0; i < no_l1; i++)
    {

        printf("\nElement : %d ", i + 1);
        insert_last(&head2, &tail2);
    }

    printf("List 1 Elements : ");
    display(head1);

    printf("\nList 2 Elements : ");
```

```
    display(head2);

    merge(&head1, &head2);
    printf("\nAfter Merge : \n");
    display(head1);
    return 0;
}
```

Output :

```
PS D:\MCA034_Aditya Nair\Termwork> cd "d:\MCA034_Aditya Nair\Termwork\" ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }

Enter number of nodes in list 1 : 5

Element : 1 Enter data for the node : 5

Element : 2 Enter data for the node : 7

Element : 3 Enter data for the node : 17

Element : 4 Enter data for the node : 13

Element : 5 Enter data for the node : 11

Enter number of nodes in list 2 : 5
Element : 1 Enter data for the node : 12
Element : 2 Enter data for the node : 10
Element : 3 Enter data for the node : 2
Element : 4 Enter data for the node : 4
Element : 5 Enter data for the node : 6
List 1 Elements :
Head -> 5 -> 7 -> 17 -> 13 -> 11 -> Tail

List 2 Elements :
Head -> 12 -> 10 -> 2 -> 4 -> 6 -> Tail

After Merge :

Head -> 5 -> 12 -> 7 -> 10 -> 17 -> 2 -> 13 -> 4 -> 11 -> 6 -> Tail
```

2 Write a program to implement stack using linked list and carry out infix to postfix conversion.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
struct node
{
    char data;
    struct node *next;
};
typedef struct node node;
node *head = NULL;
node *tail = NULL;
void push(char c)
{
```

```
    node *new_node = (node *)malloc(sizeof(node));
    if (new_node == NULL)
    {
        printf("Memory Allocation Failed.");
        exit(1);
    }
    new_node->data = c;
    new_node->next = NULL;
    if (head == NULL)
    {
        head = tail = new_node;
    }
    else
    {
        tail->next = new_node;
        tail = new_node;
    }
}
char pop()
{
    if (head == NULL)
    {
        printf("Stack is empty.");
        exit(1);
    }
    char data = tail->data;
    if (head == tail)
    {
        free(tail);
        head = tail = NULL;
    }
    else
    {
        node *temp = head;
        while (temp->next != tail)
        {
            temp = temp->next;
        }
        temp->next = NULL;
        free(tail);
        tail = temp;
    }
    return data;
}
int is_empty()
{
    return head == NULL;
}
```

```
int operator_precedence(char op)
{
   if (op == '^')
   {
      return 3;
   }
   else if (op == '*' || op == '/')
   {
      return 2;
   }
   else if (op == '+' || op == '-')
   {
      return 1;
   }
   else if (op == '(')
   {
      return 0;
   }
   else
   {
      return -1;
   }
}
void in_post()
{
   char c, postfix[100];
   int i = 0;

   printf("Enter infix expression : ");
   while ((c = getchar()) != '\n')
   {
      if (isalpha(c))
      {
         postfix[i++] = c;
      }
      else if (c == '(')
      {
         push(c);
      }
      else if (c == ')')
      {
         while (!is_empty() && tail->data != '(')
         {
            postfix[i++] = pop();
         }
         if (!is_empty() && tail->data == '(')
         {
            pop(); // Discard '('
         }
```

```
        else
        {
            printf("Mismatched parentheses.");
            exit(1);
        }
    }
    else if (c == '^' || c == '*' || c == '/' || c == '+' || c == '-')
    {
        while (!is_empty() && operator_precedence(tail->data) >= operator_precedence(c))
        {
            postfix[i++] = pop();
        }
        push(c);
    }
  }
  while (!is_empty())
  {
    if (tail->data == '(')
    {
        printf("Mismatched parentheses.");
        exit(1);
    }
    postfix[i++] = pop();
  }

  postfix[i] = '\0'; // Null terminate the string
  printf("Postfix Expression : %s\n", postfix);
}

int main()
{
  in_post();
  return 0;
}
```

Output :

```
PS D:\MCA034_Aditya Nair\Termwork> cd "d:\MCA034_Aditya Nair\Termwork\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile } ; if ($?
) { .\tempCodeRunnerFile }
Enter infix expression : (A+b)/C
Postfix Expression : Ab+C/
PS D:\MCA034_Aditya Nair\Termwork>
```

3 Write a program to find Union and Intersection of two Linked Lists [Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elements in output lists doesn't matter. Example: Input: List1: 10->15->4->20 lsit2: 8->4->2->10 Output: Intersection List: 4->10 Union List: 2->8->20->4->15->10]

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
   int data;
   struct node *next;
} *new, *head1, *head2, *tail1, *tail2, *new1;

typedef struct node node;

void create_node1(int d)
{
   new1 = (node *)malloc(1 * sizeof(node));
   if (new1 == NULL)
   {
      printf("Memory Allocation Falied.");
      return;
   }
   else
   {
      new1->next = NULL;
      new1->data = d;
   }
}

void create_node()
{
   new = (node *)malloc(1 * sizeof(node));
   if (new == NULL)
   {
      printf("Memory Allocation Falied.");
      return;
   }
   else
   {
      new->next = NULL;
      printf("Enter data for the node : ");
      scanf("%d", &new->data);
   }
}

void insert_last(node **head, node **tail)
{
```

```
    create_node();
    if (*tail == NULL)
    {
        *head = new;
        *tail = new;
    }
    else
    {
        (*tail)->next = new;
        *tail = new;
        new = NULL;
    }
}

void display(node *head)
{
    printf("\nHead -> ");
    node *temp = head;
    while (temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("Tail\n");
}

void l_union(node **head1, node **head2)
{
    node *temp1, *temp2, *nunion;
    nunion = *head1;
    temp2 = *head2;

    printf("\nUnion Result : \n");
    while (nunion->next != NULL)
    {
        printf("%d -> ", nunion->data);
        nunion = nunion->next;
    }
    printf("%d -> ", nunion->data);

    while (temp2 != NULL)
    {
        temp1 = *head1;
        int flag = 0; // Reset flag for each element in head2

        while (temp1 != NULL)
        {
            if (temp1->data == temp2->data)
            {
```

```
            flag = 1;
            break;
         }
         temp1 = temp1->next;
      }

      if (flag == 0)
      {
         create_node1(temp2->data);
         nunion->next = new1;
         nunion = nunion->next;
         printf("%d -> ", nunion->data);
      }
      temp2 = temp2->next;
   }
}

void l_intersection(node **head1, node **head2)
{
   node *temp1 = *head1;
   node *temp2;

   printf("\nIntersection Result : ");
   while (temp1 != NULL)
   {
      temp2 = *head2;
      while (temp2 != NULL)
      {
         if (temp1->data == temp2->data)
         {
            printf("%d -> ", temp1->data);
            break;
         }
         temp2 = temp2->next;
      }
      temp1 = temp1->next;
   }
   printf("NULL\n");
}




int main()
{
   int no_l1, no_l2;
   printf("\nEnter number of nodes in list 1 : ");
   scanf("%d", &no_l1);
   for (int i = 0; i < no_l1; i++)
```

```
    {
        printf("\nElement : %d ", i + 1);
        insert_last(&head1, &tail1);
    }

    printf("\nEnter number of nodes in list 2 : ");
    scanf("%d", &no_l2);
    for (int i = 0; i < no_l1; i++)
    {

        printf("\nElement : %d ", i + 1);
        insert_last(&head2, &tail2);
    }

    printf("List 1 Elements : ");
    display(head1);

    printf("\nList 2 Elements : ");
    display(head2);

    l_intersection(&head1, &head2);
    l_union(&head1, &head2);

    return 0;
}
```

Output :

```
PS D:\MCA034_Aditya Nair\Termwork> cd "d:\MCA034_Aditya Nair\Termwork\" ; if ($?) { gcc 3.c -o 3 } ; if ($?) { .\3 }

Enter number of nodes in list 1 : 4

Element : 1 Enter data for the node : 10

Element : 2 Enter data for the node : 15

Element : 3 Enter data for the node : 4

Element : 4 Enter data for the node : 20

Enter number of nodes in list 2 : 4

Element : 1 Enter data for the node : 8

Element : 2 Enter data for the node : 4

Element : 3 Enter data for the node : 2

Element : 4 Enter data for the node : 10
List 1 Elements :
Head -> 10 -> 15 -> 4 -> 20 -> Tail

List 2 Elements :
Head -> 8 -> 4 -> 2 -> 10 -> Tail

Intersection Result : 10 -> 4 -> NULL

Union Result :
10 -> 15 -> 4 -> 20 -> 8 -> 2 ->
PS D:\MCA034_Aditya Nair\Termwork>
```

4 Write a program to implement a phone directory using a singly circular linked list with following operations. Node has info like cust_id, name, phone_number.

- Insert from first
- Insert from last
- Insert at specific position
- Delete from specific position
- Delete from first
- Delete from last
- Display in sorted order
- Search by name
- Search by cust_id
- Search by phone_number
- Delete by name
- Delete by cust_id
- Delete by phone_number

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct node
{
   int cust_id;
   char name[100];
   long int phone_number;
   struct node *next;
} node;

int count = 0;

node *create(int c, char *n, long int p)
{
   node *temp = (node *)malloc(sizeof(node));
   temp->cust_id = c;
   temp->phone_number = p;
   strcpy(temp->name, n);
   temp->next = NULL;
   count++;
   return temp;
}

void insert_first(node **head, int c, char *n, long int p)
{
   node *temp = create(c, n, p);
   if (*head == NULL)
   {
      *head = temp;
```

```
      temp->next = *head;
   }
   else
   {
      temp->next = *head;
      node *last = *head;
      while (last->next != *head)
         last = last->next;
      last->next = temp;
      *head = temp;
   }
}

void insert_last(node **head, int c, char *n, long int p)
{
   node *temp = create(c, n, p);
   if (*head == NULL)
   {
      *head = temp;
      temp->next = *head;
   }
   else
   {
      node *last = *head;
      while (last->next != *head)
         last = last->next;
      last->next = temp;
      temp->next = *head;
   }
}

void delete_first(node **head)
{
   if (*head == NULL)
      return;
   node *temp = *head;
   node *last = *head;
   while (last->next != *head)
      last = last->next;
   if (last == *head)
   {
      *head = NULL;
      free(temp);
      return;
   }
   *head = temp->next;
   last->next = *head;
   free(temp);
}
```

```
void delete_last(node **head)
{
   if (*head == NULL)
      return;
   node *temp = *head;
   node *last = *head;
   while (last->next != *head)
   {
      temp = last;
      last = last->next;
   }
   if (last == *head)
   {
      *head = NULL;
      free(last);
      return;
   }
   temp->next = *head;
   free(last);
}

void insert_sorted(node **head, int c, char *n, long int p)
{
   node *temp = create(c, n, p);
   if (*head == NULL || c < (*head)->cust_id)
   {
      temp->next = *head;
      node *last = *head;
      while (last->next != *head)
         last = last->next;
      last->next = temp;
      *head = temp;
   }
   else
   {
      node *current = *head;
      while (current->next != *head && current->next->cust_id < c)
      {
         current = current->next;
      }
      temp->next = current->next;
      current->next = temp;
   }
}

void delete_from(node **head, int pos)
{
   if (*head == NULL)
```

```
        return;
    if (pos == 0)
    {
        delete_first(head);
        return;
    }
    node *prev = NULL;
    node *current = *head;
    for (int i = 0; i < pos; i++)
    {
        prev = current;
        current = current->next;
    }
    prev->next = current->next;
    count--;
    free(current);
}

void search_name(node *head, char name[100])
{
    node *temp = head;
    int h = 1;
    do
    {
        if (strcmp(temp->name, name) == 0)
        {
            printf("%d %s %ld -> ", temp->cust_id, temp->name, temp->phone_number);
            h = 0;
        }
        temp = temp->next;
    } while (temp != head);
    if (h)
    {
        printf("not found");
    }
}

void search_id(node *head, int id)
{
    node *temp = head;
    int h = 1;
    do
    {
        if (temp->cust_id == id)
        {
            printf("%d %s %ld -> ", temp->cust_id, temp->name, temp->phone_number);
            h = 0;
        }
        temp = temp->next;
```

```
    } while (temp != head);
    if (h)
    {
        printf("not found");
    }
}

void search_phoneno(node *head, long int phoneno)
{
    node *temp = head;
    int h = 1;
    do
    {
        if (temp->phone_number == phoneno)
        {
            printf("%d %s %ld -> ", temp->cust_id, temp->name, temp->phone_number);
            h = 0;
        }
        temp = temp->next;
    } while (temp != head);
    if (h)
    {
        printf("not found");
    }
}

void delete_name(node **head, char name[100])
{
    if (*head == NULL)
        return;
    node *temp = *head;
    node *prev = NULL;
    do
    {
        if (strcmp(temp->name, name) == 0)
            break;
        prev = temp;
        temp = temp->next;
    } while (temp != *head);
    if (temp == NULL || temp->next == *head)
    {
        printf("not found.\n");
        return;
    }
    if (prev == NULL)
    {
        *head = temp->next;
    }
    else
```

```
    {
        prev->next = temp->next;
    }
    count--;
    free(temp);
}

void delete_id(node **head, int id)
{
    if (*head == NULL)
        return;
    node *temp = *head;
    node *prev = NULL;
    do
    {
        if (temp->cust_id == id)
            break;
        prev = temp;
        temp = temp->next;
    } while (temp != *head);
    if (temp == NULL || temp->next == *head)
    {
        printf("not found.\n");
        return;
    }
    if (prev == NULL)
    {
        *head = temp->next;
    }
    else
    {
        prev->next = temp->next;
    }
    count--;
    free(temp);
}

void delete_phoneno(node **head, long int phoneno)
{
    if (*head == NULL)
        return;
    node *temp = *head;
    node *prev = NULL;
    do
    {
        if (temp->phone_number == phoneno)
            break;
        prev = temp;
        temp = temp->next;
```

```
    } while (temp != *head);
    if (temp == NULL || temp->next == *head)
    {
        printf("not found.\n");
        return;
    }
    if (prev == NULL)
    {
        *head = temp->next;
    }
    else
    {
        prev->next = temp->next;
    }
    count--;
    free(temp);
}

void display(node *head)
{
    if (head == NULL)
    {
        printf("empty.\n");
        return;
    }

    node *current = head;
    node *index = NULL;
    int temp_id;
    char temp_name[100];
    long int temp_phone;

    do
    {
        index = current->next;

        while (index != head)
        {
            if (current->cust_id > index->cust_id)
            {
                temp_id = current->cust_id;
                current->cust_id = index->cust_id;
                index->cust_id = temp_id;

                strcpy(temp_name, current->name);
                strcpy(current->name, index->name);
                strcpy(index->name, temp_name);

                temp_phone = current->phone_number;
```

```
            current->phone_number = index->phone_number;
            index->phone_number = temp_phone;
        }
        index = index->next;
      }
      current = current->next;
   } while (current->next != head);
   current = head;
   do
   {
      printf("%d %s %ld -> ", current->cust_id, current->name, current->phone_number);
      current = current->next;
   } while (current != head);
   printf("\n");
}


int main()
{
   node *head = NULL;
   int choice = 1;
   while (choice != 0)
   {
      int c;
      char n[100];
      long int p;
      printf("\n1. insert from first\n2. insert from last\n3. insert at directory sorting position based on cust id\n4. delete
from specific position\n5. delete from first\n6. delete from last\n7. display \n8. search by name\n9. search by cust
id\n10. search by phone number\n11. delete by name\n12. delete by cust id\n13. delete by phone number\n0.exit\n");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         scanf("%d %s %ld", &c, n, &p);
         insert_first(&head, c, n, p);
         break;
      case 2:
         scanf("%d %s %ld", &c, n, &p);
         insert_last(&head, c, n, p);
         break;
      case 3:
         scanf("%d %s %ld", &c, n, &p);
         insert_sorted(&head, c, n, p);
         break;
      case 4:
         printf("enter position to delete: ");
         scanf("%d", &c);
         delete_from(&head, c);
         break;
```

```
      case 5:
         delete_first(&head);
         break;
      case 6:
         delete_last(&head);
         break;
      case 7:
         display(head);
         break;
      case 8:
         printf("enter name: ");
         scanf("%s", n);
         search_name(head, n);
         break;
      case 9:
         printf("enter customer id: ");
         scanf("%d", &c);
         search_id(head, c);
         break;
      case 10:
         printf("enter phone number: ");
         scanf("%ld", &p);
         search_phoneno(head, p);
         break;
      case 11:
         printf("enter name : ");
         scanf("%s", n);
         delete_name(&head, n);
         break;
      case 12:
         printf("enter customer id : ");
         scanf("%d", &c);
         delete_id(&head, c);
         break;
      case 13:
         printf("enter phone number : ");
         scanf("%ld", &p);
         delete_phoneno(&head, p);
         break;
      case 0:
      default:
         break;
      }
   }
   return 0;
}
```

Output :

```
PS D:\MCA034_Aditya Nair\Termwork> cd "d:\MCA034_Aditya Nair\Termwork\" ; if ($?) { gcc 4.c -o 4 } ; if ($?) { .\4 }

1. insert from first
2. insert from last
3. insert at directory sorting position based on cust id
4. delete from specific position
5. delete from first
6. delete from last
7. display
8. search by name
9. search by cust id
10. search by phone number
11. delete by name
12. delete by cust id
13. delete by phone number
0.exit
1
1 aditya 123456789

1. insert from first
2. insert from last
3. insert at directory sorting position based on cust id
4. delete from specific position
5. delete from first
6. delete from last
7. display
8. search by name
9. search by cust id
10. search by phone number
11. delete by name
12. delete by cust id
13. delete by phone number
0.exit
```

5 Write a program to implement priority queue using linked list.

```
#include <stdio.h>
#include<stdlib.h>
typedef struct node
{
   int data;
   struct node *next;
}node;
int count = 0;

node *create(int data)
{
   node *temp = (node *)malloc(sizeof(node));
   temp->data= data;
   temp->next = NULL;
   count++;
   return temp;
}

void insert (node **head, int data)
{
   node *temp = create(data);
   if(*head == NULL)
   {
      *head = temp;
   }
```

```
    else
    {
        temp->next = *head;
        *head = temp;
    }
}

void delete_last(node **head)
{
    node *prev = NULL;
    node *current = *head;
    while(current->next != NULL)
    {
        prev = current;
        current = current->next;
    }
    prev->next = NULL;
    count--;
}

void display(node *head)
{
    node *temp = head;
    while(temp !=NULL)
    {
        printf("%d -> ",temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void priority_enqueue(node **head, int data)
{
    insert(head, data);
    node *temp1 = *head;
    node *temp2 = NULL;
    while (temp1 != NULL)
    {
        temp2 = temp1->next;
        while (temp2 != NULL)
        {
            if (temp1->data > temp2->data)
            {
                int h = temp1->data;
                temp1->data = temp2->data;
                temp2->data = h;
            }
            temp2 = temp2->next;
        }
```

```
        temp1 = temp1->next;
    }
    printf("\n");
}

void priority_dequeue(node **head)
{
    delete_last(head);
}

int main()
{
    node *head = NULL;
    priority_enqueue(&head, 50);
    priority_enqueue(&head, 26);
    priority_enqueue(&head, 1);
    priority_enqueue(&head, 20);
    priority_dequeue(&head);
    display(head);
    return 0;
}
```

Output :

```
PS D:\MCA034_Aditya Nair\Termwork> cd "d:\MCA034_Aditya Nair\Termwork\" ; if ($?) { gcc 5.c -o 5 } ; if ($?) { .\5 }



1 -> 20 -> 26 ->
PS D:\MCA034_Aditya Nair\Termwork>
```

6 For this program, you will generate two different types of graphs and compute using them. [Generate from provided two files]

File format

- Input will be based on file.
- Assume vertices are numbered 0..n-1.
- In this case, we will assume each file contains exactly one graph.
- Every graph has a two line "header".
  - Line 1: isDirected isWeighted
  - Line 2: n m On line 1, if isDirected==0 the graph is undirected, else it is directed. If isWeighted==0 the graph is unweighted else it is weighted. On line 2, n is the number of vertices (nodes) and m is the number of edges.

The next m lines contain information about the edges. If the graph isWeighted, the next m lines each contain three integers, u, v and w, where u and v are the endpoints of the edge and w is the weight on that edge. If the graph is not weighted, each of the m lines contains only u and v. If the graph is undirected, there is an edge (u, v) and an edge (v, u). If the graph isDirected, the edge is from u to v.

```
#include <stdio.h>
```

```
#include <stdlib.h>

#define MAX 100

typedef struct
{
   int isDirected;
   int isWeighted;
   int n;
   int m;
   int adjMatrix[MAX][MAX];
   int adjList[MAX][MAX];
} Graph;

Graph *createGraph(int isDirected, int isWeighted, int n, int m)
{
   Graph *graph = (Graph *)malloc(sizeof(Graph));
   graph->isDirected = isDirected;
   graph->isWeighted = isWeighted;
   graph->n = n;
   graph->m = m;
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         graph->adjMatrix[i][j] = 0;
         graph->adjList[i][j] = 0;
      }
   }
   return graph;
}

void addEdge(Graph *graph, int u, int v, int w)
{
   if (graph->isDirected)
   {
      graph->adjMatrix[u][v] = w;
      graph->adjList[u][v] = w;
   }
   else
   {
      graph->adjMatrix[u][v] = graph->adjMatrix[v][u] = w;
      graph->adjList[u][v] = graph->adjList[v][u] = w;
   }
}

void dfs(Graph *graph, int vertex, int visited[])
{
   visited[vertex] = 1;
```

```
    printf("%d ", vertex);
    for (int i = 0; i < graph->n; i++)
    {
        if (graph->adjList[vertex][i] && !visited[i])
        {
            dfs(graph, i, visited);
        }
    }
}

void bfs(Graph *graph, int vertex)
{
    int queue[MAX], front = 0, rear = -1, visited[MAX] = {0};
    queue[++rear] = vertex;
    visited[vertex] = 1;
    while (front <= rear)
    {
        vertex = queue[front++];
        printf("%d ", vertex);
        for (int i = 0; i < graph->n; i++)
        {
            if (graph->adjList[vertex][i] && !visited[i])
            {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
}

int main()
{
    FILE *file1 = fopen("tgraph.txt", "r");
    int isDirected1, isWeighted1, n1, m1;
    fscanf(file1, "%d %d\n%d %d", &isDirected1, &isWeighted1, &n1, &m1);
    Graph *graph1 = createGraph(isDirected1, isWeighted1, n1, m1);
    for (int i = 0; i < m1; i++)
    {
        int u, v, w = 1;
        if (isWeighted1)
        {
            fscanf(file1, "%d %d %d", &u, &v, &w);
        }
        else
        {
            fscanf(file1, "%d %d", &u, &v);
        }
        addEdge(graph1, u, v, w);
    }
```

```
    fclose(file1);

    FILE *file2 = fopen("fgraph.txt", "r");
    int isDirected2, isWeighted2, n2, m2;
    fscanf(file2, "%d %d\n%d %d", &isDirected2, &isWeighted2, &n2, &m2);
    Graph *graph2 = createGraph(isDirected2, isWeighted2, n2, m2);
    for (int i = 0; i < m2; i++)
    {
        int u, v, w = 1;
        if (isWeighted2)
        {
            fscanf(file2, "%d %d %d", &u, &v, &w);
        }
        else
        {
            fscanf(file2, "%d %d", &u, &v);
        }
        addEdge(graph2, u, v, w);
    }
    fclose(file2);

    printf("DFS traversal of tgraph.txt: ");
    int visited1[MAX] = {0};
    dfs(graph1, 0, visited1);
    printf("\nBFS traversal of tgraph.txt: ");
    bfs(graph1, 0);

    printf("\n\nDFS traversal of fgraph.txt: ");
    int visited2[MAX] = {0};
    dfs(graph2, 0, visited2);
    printf("\nBFS traversal of fgraph.txt: ");
    bfs(graph2, 0);

    return 0;
}
```

Output :

```
PS D:\MCA034_Aditya Nair\Termwork> cd "d:\MCA034_Aditya Nair\Termwork\" ; if ($?) { gcc 6.c -o 6 } ; if ($?) { .\6 }
DFS traversal of tgraph.txt: 0 1 3 4 5 7 6 2
BFS traversal of tgraph.txt: 0 1 2 6 7 3 4 5

DFS traversal of fgraph.txt: 0 1 2 3 8 4 9 10 5 12 6 7 11
BFS traversal of fgraph.txt: 0 1 2 3 8 4 6 9 11 7 10 5 12
PS D:\MCA034_Aditya Nair\Termwork>
```