Dharmsinh Desai University

Academic Year 2022-23

Department:

Faculty of Management and information science

Subject:

Data Structures

Full Name: Sutariya Savankumar Sureshbhai

Roll No.: MA065

ID No.: 22MAPOG030

Submitted to: Prof. Himanshu K Purohit | MCA Department

Student sign. **Professor sign.**

1. **Write a program to Append Last N Nodes to First in the Linked List [Given a linked list and an integer n, append the last n elements of the LL to front. Assume given n will be smaller than length of LL. [Input format: Line 1: Linked list elements (separated by space and terminated by -1**
   Sample Input 1 : 1 2 3 4 5 -1
   3
   Sample Output 1 : 3 4 5 1 2

Code (intlist.h)

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
    int val;
    struct node *next;
}node;
typedef struct linklist{
    node *head;
}linklist;
node *createNode(int val){
    node *new = (node *)malloc(sizeof(node));
    new->val = val;
    new->next = NULL;
    return new;
}
void insert(linklist *l,int val){
    node *new = createNode(val);
    if(l->head == NULL){
        l->head = new;
        return;
    }
    node *tmp = l->head;
    while(tmp->next != NULL){
        tmp = tmp->next;
```

```
    }
    tmp->next = new;
}
void display(linklist *l){
    printf("\n");
    if(l->head == NULL){
        printf("Empty");
        return;
    }
    node *tmp = l->head;
    while(tmp != NULL){
        printf("%d ",tmp->val);
        tmp = tmp->next;
    }
}
```

Code(p1.c)

```
#include<stdio.h>
#include<stdlib.h>
#include"intlist.h"
void appendToFirstFromLast(linklist *l){
    if(l->head == NULL || l->head->next == NULL){
        return;
    }
    node *tmp = l->head;
    while(tmp->next->next != NULL){
        tmp = tmp->next;
    }
    tmp->next->next = l->head;
    l->head = tmp->next;
    tmp->next = NULL;
}
int main(){
    linklist l1;
    l1.head = NULL;
    while(1){
        int val;
        scanf("%d",&val);
        if(val == -1)
            break;
```

```
        insert(&l1,val);
    }
    int n;
    scanf("%d",&n);
    for(int i=0; i<n; i++)
        appendToFirstFromLast(&l1);
    display(&l1);
    return 0;
}
```

Output

```
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> gcc .\p1.c -o p1
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> ./p1
1 2 3 4 5 -1
3

3 4 5 1 2
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> ./p1
1 2 3 4 5 6 -1
5

2 3 4 5 6 1
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> ./p1
1 2 3 -1
2

2 3 1
```

2. Write a program to implement stack using linked list which converts infix to postfix.

Code

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct node{
    char val;
    struct node *next;
}node;
typedef struct linklist{
    node *head;
}linklist;

node *createNode(char val){
    node *new = (node *)malloc(sizeof(node));
    new->val = val;
```

```
    new->next = NULL;
    return new;
}

void push(linklist *l,char val){
    node *new = createNode(val);
    if(l->head == NULL){
        l->head = new;
        return;
    }
    node *tmp = l->head;
    while(tmp->next != NULL){
        tmp = tmp->next;
    }
    tmp->next = new;
}

char pop(linklist *l){
    if(l->head == NULL){
        printf("Empty");
        return -1;
    }
    node *tmp = l->head;
    char val = tmp->val;
    l->head = tmp->next;
    free(tmp);
    return val;
}

char peek(linklist *l){
    if(l->head == NULL){
        printf("Empty");
        return -1;
    }
    node *tmp = l->head;
    char val = tmp->val;
    return val;
}

int isEmpty(linklist *l){
    if(l->head == NULL){
        return 1;
    }
    return 0;
}

int isOperand(char c){
    if((c >= 'a' && c <= 'z' ) || (c >= 'A' && c <= 'Z' )){
```

```
        return 1;
    }
    return 0;
}

int precedence(char c){
    if(c == '^'){
        return 3;
    }
    else if(c == '*' || c == '/'){
        return 2;
    }
    else if(c == '+' || c == '-'){
        return 1;
    }
    return -1;
}

void infixToPostfix(char *exp){
    linklist l;
    l.head = NULL;
    int i,k;
    for(i = 0,k = -1;exp[i] != '\0';i++){
        if(isOperand(exp[i])){
            exp[++k] = exp[i];
        }
        else if(exp[i] == '('){
            push(&l,exp[i]);
        }
        else if(exp[i] == ')'){
            while(!isEmpty(&l) && peek(&l) != '('){
                exp[++k] = pop(&l);
            }
            if(!isEmpty(&l) && peek(&l) != '('){
                printf("Invalid Expression");
                return;
            }
            else
                pop(&l);
        }
        else{
            while(!isEmpty(&l) && precedence(exp[i]) <= precedence(peek(&l))){
                exp[++k] = pop(&l);
            }
            push(&l,exp[i]);
        }
    }
    while(!isEmpty(&l)){
```

```
        exp[++k] = pop(&l);
    }
    exp[++k] = '\0';
    printf("%s",exp);
}

int main(){
    char exp[100];
    scanf("%s",exp);
    infixToPostfix(exp);
    return 0;
}
```

Output

```
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> gcc .\p2.c -o p2
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> ./p2
a+b-(b+c*a)
ab+b-ca+*
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution>
```

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**

Code

```
#include<stdio.h>
#include<stdlib.h>
#include"intlist.h"
void interUnion(linklist *l1, linklist *l2, linklist *i, linklist *u){
    if(l1->head == NULL && l2->head == NULL)
        return;
    node *tmp1 = l1->head;
    //first list + intersacting into union
    while(tmp1!=NULL){
        node *tmp2 = l2->head;
        while(tmp2!=NULL){
            if(tmp1->val == tmp2->val)
                insert(i,tmp1->val);
            tmp2=tmp2->next;
        }
```

```
        insert(u,tmp1->val);
        tmp1=tmp1->next;
    }
    //remaining union from list2
    tmp1 = l2->head;
    while(tmp1 != NULL){
        node *tmp2 = i->head;
        short flag = 0;
        while(tmp2 != NULL){
            if(tmp1->val == tmp2->val){
                flag=1;
                break;
            }
            tmp2 = tmp2->next;
        }
        if(!flag)
            insert(u,tmp1->val);
        tmp1 = tmp1->next;
    }
}
int main(){
    linklist l1,l2,unionlist,interlist;
    l1.head = NULL;
    l2.head = NULL;
    unionlist.head = NULL;
    interlist.head = NULL;
    while(1){
        int n;
        scanf("%d",&n);
        if(n==-1)
            break;
        insert(&l1,n);
    }
    while(1){
        int n;
        scanf("%d",&n);
        if(n==-1)
            break;
        insert(&l2,n);
    }
    interUnion(&l1,&l2,&interlist,&unionlist);
    printf("intersection list : ");
    display(&interlist);
    printf("\nunion list : ");
    display(&unionlist);
    return 0;
}
```

Output

```
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> gcc .\p3.c -o p3
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> ./p3
10 15 4 20 -1
8 4 2 10 -1
intersection list :
10 4
union list :
10 15 4 20 8 2
```

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 directory sorting position based on cust_id**
   - **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**

Code

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

typedef struct contact{
    int cust_id;
    char name[20];
    char phone_number[13];
    struct contact *next;
}contact;

typedef struct directory{
    contact *head;
}directory;
```

```
contact *createContact(int cust_id,char name[],char phone_number[]){
    contact *new = (contact *)malloc(sizeof(contact));
    new->cust_id = cust_id;
    strcpy(new->name,name);
    strcpy(new->phone_number,phone_number);
    new->next = NULL;
    return new;
}
// Check if already exists
int exists(directory *d,int cust_id){
    if(d->head == NULL){
        return 0;
    }
    contact *tmp = d->head;
    while(tmp->next != d->head){
        if(tmp->cust_id == cust_id){
            return 1;
        }
        tmp = tmp->next;
    }
    if(tmp->cust_id == cust_id){
        return 1;
    }
    return 0;
}
// insert at first
void insertAtFirst(directory *d,int cust_id,char name[],char phone_number[]){
    if(exists(d,cust_id)){
        printf("Already exists");
        return;
    }
    contact *new = createContact(cust_id,name,phone_number);
    if(d->head == NULL){
        d->head = new;
        new->next = new;
        return;
    }
    contact *tmp = d->head;
    while(tmp->next != d->head){
        tmp = tmp->next;
    }
    tmp->next = new;
    new->next = d->head;
    d->head = new;
}
// insert at last
void insertAtLast(directory *d,int cust_id,char name[],char phone_number[]){
    if(exists(d,cust_id)){
        printf("Already exists");
        return;
    }
    contact *new = createContact(cust_id,name,phone_number);
```

```
    if(d->head == NULL){
        d->head = new;
        new->next = new;
        return;
    }
    contact *tmp = d->head;
    while(tmp->next != d->head){
        tmp = tmp->next;
    }
    tmp->next = new;
    new->next = d->head;
}
// insert in order of cust_id
void insertInOrder(directory *d,int cust_id,char name[],char phone_number[]){

    contact *new = createContact(cust_id,name,phone_number);
    if(d->head == NULL){
        d->head = new;
        new->next = new;
        return;
    }
    contact *tmp = d->head;
    if(tmp->cust_id > cust_id){
        while(tmp->next != d->head){
            tmp = tmp->next;
        }
        tmp->next = new;
        new->next = d->head;
        d->head = new;
        return;
    }
    while(tmp->next != d->head && tmp->next->cust_id < cust_id){
        tmp = tmp->next;
    }
    new->next = tmp->next;
    tmp->next = new;
}
// delete first
void deleteFirst(directory *d){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
    while(tmp->next != d->head){
        tmp = tmp->next;
    }
    tmp->next = d->head->next;
    tmp = d->head;
    d->head = d->head->next;
    free(tmp);
}
```

```
// delete last
void deleteLast(directory *d){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
    while(tmp->next->next != d->head){
        tmp = tmp->next;
    }
    contact *tmp2 = tmp->next;
    tmp->next = d->head;
    free(tmp2);
}
// delete from specific
void deleteFromSpecific(directory *d,int cust_id){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
    if(tmp->cust_id == cust_id){
        while(tmp->next != d->head){
            tmp = tmp->next;
        }
        tmp->next = d->head->next;
        tmp = d->head;
        d->head = d->head->next;
        free(tmp);
        return;
    }
    while(tmp->next != d->head && tmp->next->cust_id != cust_id){
        tmp = tmp->next;
    }
    if(tmp->next == d->head){
        printf("Not found");
        return;
    }
    contact *tmp2 = tmp->next;
    tmp->next = tmp->next->next;
    free(tmp2);
}

// Search by name ----------------------------------------
void searchByName(directory *d,char name[]){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
    while(tmp->next != d->head){
        if(strcmp(tmp->name,name) == 0){
```

```
            printf("Found\n");
            printf("Customer ID: %d\n",tmp->cust_id);
            printf("Name: %s\n",tmp->name);
            printf("Phone Number: %s\n",tmp->phone_number);
            return;
        }
        tmp = tmp->next;
    }
    if(strcmp(tmp->name,name) == 0){
        printf("Found\n");
        printf("Customer ID: %d\n",tmp->cust_id);
        printf("Name: %s\n",tmp->name);
        printf("Phone Number: %s\n",tmp->phone_number);
        return;
    }
    printf("Not found");
}

// Search by cust_id ----------------------------------------
void searchByCustId(directory *d,int cust_id){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
    while(tmp->next != d->head){
        if(tmp->cust_id == cust_id){
            printf("Found\n");
            printf("Customer ID: %d\n",tmp->cust_id);
            printf("Name: %s\n",tmp->name);
            printf("Phone Number: %s\n",tmp->phone_number);
            return;
        }
        tmp = tmp->next;
    }
    if(tmp->cust_id == cust_id){
        printf("Found\n");
        printf("Customer ID: %d\n",tmp->cust_id);
        printf("Name: %s\n",tmp->name);
        printf("Phone Number: %s\n",tmp->phone_number);
        return;
    }
    printf("Not found");
}

// Search by phone number -----------------------------------------
void searchByPhoneNumber(directory *d,char phone_number[]){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
```

```
    while(tmp->next != d->head){
        if(strcmp(tmp->phone_number,phone_number) == 0){
            printf("Found\n");
            printf("Customer ID: %d\n",tmp->cust_id);
            printf("Name: %s\n",tmp->name);
            printf("Phone Number: %s\n",tmp->phone_number);
            return;
        }
        tmp = tmp->next;
    }
    if(strcmp(tmp->phone_number,phone_number) == 0){
        printf("Found\n");
        printf("Customer ID: %d\n",tmp->cust_id);
        printf("Name: %s\n",tmp->name);
        printf("Phone Number: %s\n",tmp->phone_number);
        return;
    }
    printf("Not found");
}

// delete by name
void deleteByName(directory *d,char name[]){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
    if(strcmp(tmp->name,name) == 0){
        while(tmp->next != d->head){
            tmp = tmp->next;
        }
        tmp->next = d->head->next;
        tmp = d->head;
        d->head = d->head->next;
        free(tmp);
        return;
    }
    while(tmp->next != d->head && strcmp(tmp->next->name,name) != 0){
        tmp = tmp->next;
    }
    if(tmp->next == d->head){
        printf("Not found");
        return;
    }
    contact *tmp2 = tmp->next;
    tmp->next = tmp->next->next;
    free(tmp2);
}

// delete by cust_id
void deleteByCustId(directory *d,int cust_id){
    if(d->head == NULL){
```

```
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
    if(tmp->cust_id == cust_id){
        while(tmp->next != d->head){
            tmp = tmp->next;
        }
        tmp->next = d->head->next;
        tmp = d->head;
        d->head = d->head->next;
        free(tmp);
        return;
    }
    while(tmp->next != d->head && tmp->next->cust_id != cust_id){
        tmp = tmp->next;
    }
    if(tmp->next == d->head){
        printf("Not found");
        return;
    }
    contact *tmp2 = tmp->next;
    tmp->next = tmp->next->next;
    free(tmp2);
}

// delete by phone number
void deleteByPhoneNumber(directory *d,char phone_number[]){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    contact *tmp = d->head;
    if(strcmp(tmp->phone_number,phone_number) == 0){
        while(tmp->next != d->head){
            tmp = tmp->next;
        }
        tmp->next = d->head->next;
        tmp = d->head;
        d->head = d->head->next;
        free(tmp);
        return;
    }
    while(tmp->next != d->head && strcmp(tmp->next->phone_number,phone_number) != 0){
        tmp = tmp->next;
    }
    if(tmp->next == d->head){
        printf("Not found");
        return;
    }
    contact *tmp2 = tmp->next;
    tmp->next = tmp->next->next;
```

```
    free(tmp2);
}

// display
void display(directory *d){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    printf("\n--------------Directory--------------");
    contact *tmp = d->head;
    while(tmp->next != d->head){
        printf("\n%d",tmp->cust_id);
        printf(" %20s",tmp->name);
        printf(" %13s",tmp->phone_number);
        tmp = tmp->next;
    }
    printf("\n%d",tmp->cust_id);
    printf(" %20s",tmp->name);
    printf(" %13s",tmp->phone_number);
    printf("\n-------------------------------------");

}

// display sorted
void displaySorted(directory *d){
    // copy the list then sort that
    directory d1;
    d1.head = NULL;
    contact *tmp = d->head;
    while(tmp->next != d->head){
        insertInOrder(&d1,tmp->cust_id,tmp->name,tmp->phone_number);
        tmp = tmp->next;
    }
    insertInOrder(&d1,tmp->cust_id,tmp->name,tmp->phone_number);
    display(&d1);
}

int main(){
    // menu
    directory d;
    d.head = NULL;
    int id;
    char name[20];
    char phone_number[13];
    while(1){
        printf("\n1. Insert at first\n");
        printf("2. Insert at last\n");
        printf("3. Insert in sorted order\n");
        printf("4. Delete from first\n");
        printf("5. Delete from last\n");
        printf("6. Delete from specific position\n");
```

```
        printf("7. Display in sorted order\n");
        printf("8. Search by name\n");
        printf("9. Search by cust_id\n");
        printf("10. Search by phone_number\n");
        printf("11. Delete by name\n");
        printf("12. Delete by cust_id\n");
        printf("13. Delete by phone_number\n");
        printf("14. Display\n");
        printf("15. Exit\n");

        printf("Enter your choice: ");
        int ch;
        scanf("%d",&ch);
        switch(ch){
            case 1:
                printf("Enter cust_id: ");
                scanf("%d",&id);
                printf("Enter name: ");
                scanf("%s",name);
                printf("Enter phone_number: ");
                scanf("%s",phone_number);
                insertAtFirst(&d,id,name,phone_number);
                break;
            case 2:
                printf("Enter cust_id: ");
                scanf("%d",&id);
                printf("Enter name: ");
                scanf("%s",name);
                printf("Enter phone_number: ");
                scanf("%s",phone_number);
                insertAtLast(&d,id,name,phone_number);
                break;
            case 3:
                printf("The sequence of directory may get changed");
                printf("Do you want to continue (y/n): ");
                char ch;
                scanf(" %c",&ch);
                if(ch == 'n')
                    break;
                printf("Enter cust_id: ");
                scanf("%d",&id);
                printf("Enter name: ");
                scanf("%s",name);
                printf("Enter phone_number: ");
                scanf("%s",phone_number);
                insertInOrder(&d,id,name,phone_number);
                break;
            case 4:
                deleteFirst(&d);
                break;
            case 5:
                deleteLast(&d);
```

```
                break;
            case 6:
                printf("Enter position: ");
                int pos;
                scanf("%d",&pos);
                deleteFromSpecific(&d,pos);
                break;
            case 7:
                displaySorted(&d);
                break;
            case 8:
                printf("Enter name: ");
                scanf("%s",name);
                searchByName(&d,name);
                break;
            case 9:
                printf("Enter cust_id: ");
                scanf("%d",&id);
                searchByCustId(&d,id);
                break;
            case 10:
                printf("Enter phone_number: ");
                scanf("%s",phone_number);
                searchByPhoneNumber(&d,phone_number);
                break;
            case 11:
                printf("Enter name: ");
                scanf("%s",name);
                deleteByName(&d,name);
                break;
            case 12:
                printf("Enter cust_id: ");
                scanf("%d",&id);
                deleteByCustId(&d,id);
                break;
            case 13:
                printf("Enter phone_number: ");
                scanf("%s",phone_number);
                deleteByPhoneNumber(&d,phone_number);
                break;
            case 14:
                display(&d);
                break;
            case 15:
                exit(0);
            default:
                printf("Invalid choice");
        }
    }
    return 0;
}
```

Output

```
--------------Directory--------------
11                 savan    3232455434
10                 Mayur    3211237654
14                Kairav    3212448524
12                Sanmay    2332125776
17                 Yagna    2422340345
15                   Mit    2144325445
-------------------------------------
1. Insert at first
2. Insert at last
3. Insert in sorted order
4. Delete from first
5. Delete from last
6. Delete from specific position
7. Display in sorted order
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
14. Display
15. Exit
Enter your choice:
```

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

Code

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
    int val;
    int priority;
    struct node *next;
}node;
typedef struct linklist{
    node *head;
}linklist;

node *create(int val,int priority){
    node *new = (node *)malloc(sizeof(node));
```

```
    new->val = val;
    new->priority = priority;
    new->next = NULL;
    return new;
}
//insert in priority order
void enqueue(linklist *l,int val,int priority){
    node *new = create(val,priority);
    if(l->head == NULL){
        l->head = new;
        return;
    }
    node *tmp = l->head;
    if(tmp->priority > priority){
        new->next = tmp;
        l->head = new;
        return;
    }
    while(tmp->next != NULL && tmp->next->priority <= priority){
        tmp = tmp->next;
    }
    new->next = tmp->next;
    tmp->next = new;
}
//delete from front
int dequeue(linklist *l){
    if(l->head == NULL){
        printf("Empty");
        return -1;
    }
    node *tmp = l->head;
    int val = tmp->val;
    l->head = tmp->next;
    free(tmp);
    return val;
}

void display(linklist *l){
    printf("\n");
    if(l->head == NULL){
        printf("Empty");
        return;
    }
    node *tmp = l->head;
    while(tmp != NULL){
        printf("%d ",tmp->val);
        tmp = tmp->next;
    }
```

```
}

int main(){
    // menu
    linklist l;
    l.head = NULL;
    int ch,val,priority;
    while(1){
        printf("\n1.Enqueue\n");
        printf("2.Dequeue\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice: ");
        scanf("%d",&ch);
        switch(ch){
            case 1:
                printf("Enter value and priority: ");
                scanf("%d%d",&val,&priority);
                enqueue(&l,val,priority);
                break;
            case 2:
                printf("Removed %d\n",dequeue(&l));
                break;
            case 3:
                display(&l);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice");
        }
    }
}
```

Output

```
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> gcc .\p5.c -o p5
PS C:\Drive\Study\MCA\DDU\SEM_2\DS\Termwork\Solution> ./p5

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 1
Enter value and priority: 1 5

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 1
Enter value and priority: 2
3

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 1
Enter value and priority: 3
7

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 2
Removed 2

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 4
```

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

Code

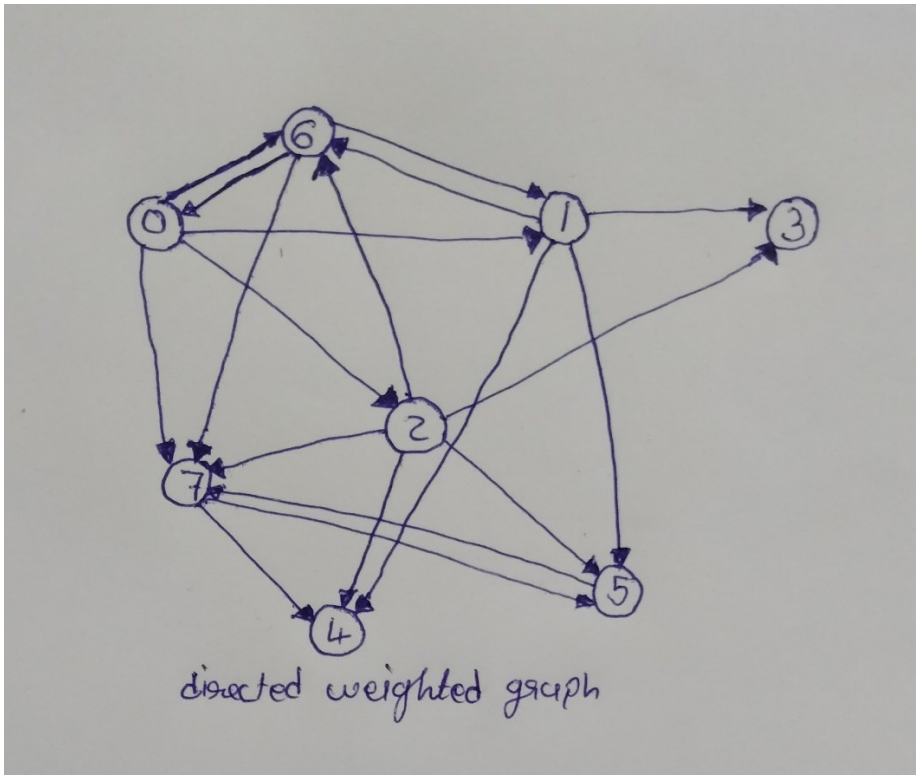
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODES 100
int n, m, isDirected, isWeighted;
int adjMatrix[MAX_NODES][MAX_NODES] = {0};
int adjList[MAX_NODES][MAX_NODES] = {0};
void dfs(int v, int visited[]) {
    visited[v] = 1;
    printf("%d ", v);
    for (int i = 0; i < n; i++) {
        if (adjMatrix[v][i] && !visited[i]) {
            dfs(i, visited);
        }
    }
}
void bfs(int start) {
    int visited[MAX_NODES] = {0};
    int queue[MAX_NODES], front = 0, rear = 0;
    visited[start] = 1;
    printf("%d ", start);
    queue[rear++] = start;
    while (front < rear) {
        int v = queue[front++];
        for (int i = 0; i < n; i++) {
            if (adjMatrix[v][i] && !visited[i]) {
                visited[i] = 1;
                printf("%d ", i);
                queue[rear++] = i;
            }
        }
    }
}
void printAdjMatrix() {
    printf("\nAdjacency Matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", adjMatrix[i][j]);
        }
        printf("\n");
    }
}
void generateAdjMatrix(FILE *fp) {
    int u, v, w;
    for (int i = 0; i < m; i++) {
        if (isWeighted) {
            fscanf(fp, "%d %d %d", &u, &v, &w);
        } else {
```

```
            fscanf(fp, "%d %d", &u, &v);
        }
        adjMatrix[u][v] = 1;
        if (!isDirected) {
            adjMatrix[v][u] = 1;
        }
    }
    printAdjMatrix();
}
struct Node {
    int dest;
    int weight;
    struct Node* next;
};
struct Graph {
    int V;
    struct Node** adjList;
};
struct Node* createNode(int dest, int weight) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->dest = dest;
    newNode->weight = weight;
    newNode->next = NULL;
    return newNode;
}
struct Graph* createGraph(int V) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->V = V;
    graph->adjList = (struct Node**)malloc(V * sizeof(struct Node*));
    for (int i = 0; i < V; i++) {
        graph->adjList[i] = NULL;
    }
    return graph;
}
void addEdge(struct Graph* graph, int src, int dest, int weight) {
    struct Node* newNode = createNode(dest, weight);
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;
    if (!isDirected) {
        newNode = createNode(src, weight);
        newNode->next = graph->adjList[dest];
        graph->adjList[dest] = newNode;
    }
}
void printAdjList(struct Graph *graph) {
    printf("\nAdjacency List:\n");
    for (int i = 0; i < graph->V; i++) {
        struct Node* temp = graph->adjList[i];
        printf("Vertex %d: ", i);
        while (temp) {
            printf("%d ", temp->dest);
            temp = temp->next;
```

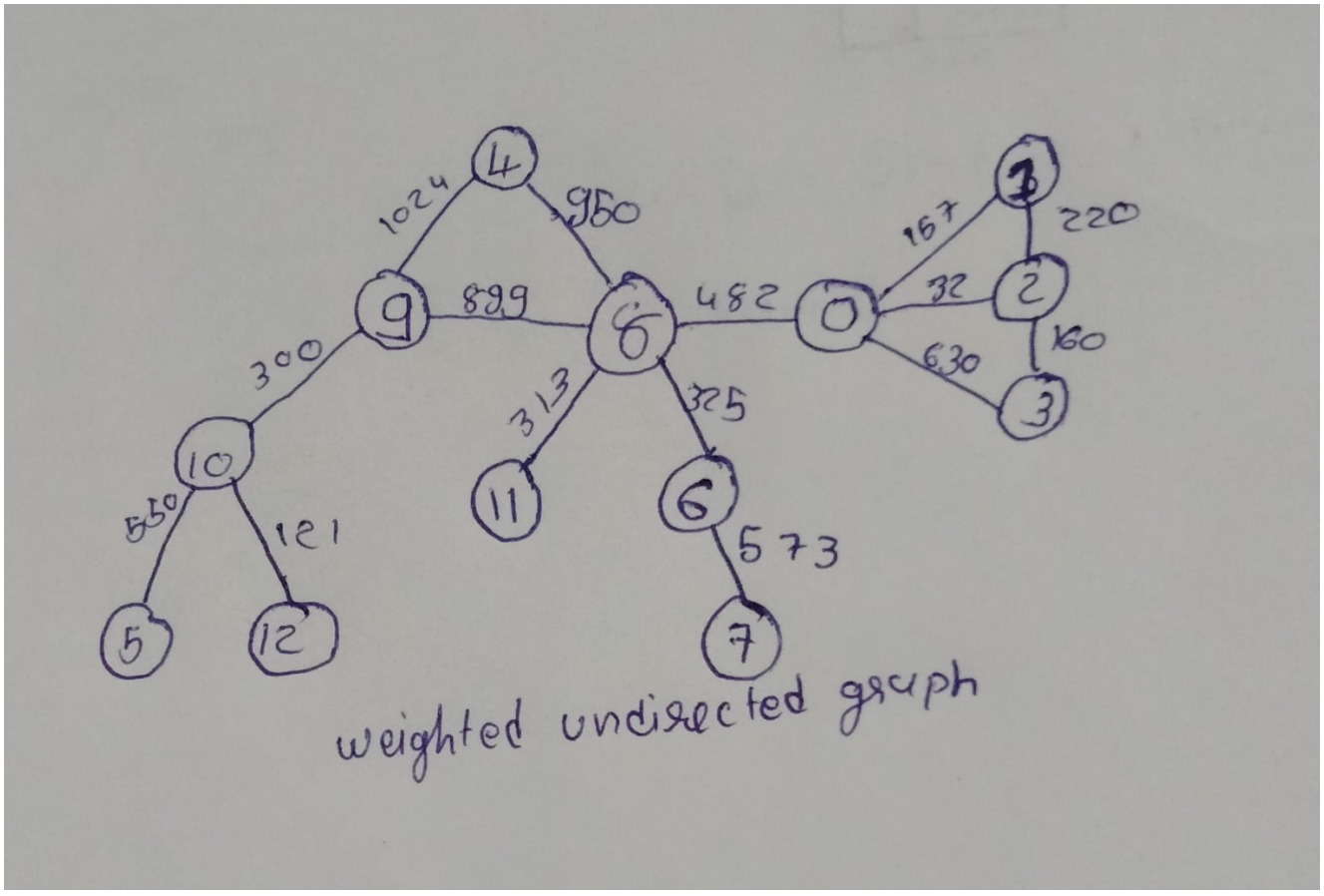
```
        }
        printf("\n");
    }
}
void generateAdjList(FILE *fp) {
    int u, v, w;
    struct Graph* graph = createGraph(n);
    for (int i = 0; i < m; i++) {
        if (isWeighted) {
            fscanf(fp, "%d %d %d", &u, &v, &w);
        } else {
            fscanf(fp, "%d %d", &u, &v);
        }
        addEdge(graph, u, v, w);
    }
    printAdjList(graph);
}
int main() {
    FILE *fp;
    fp = fopen("tgraph.txt", "r");
    fscanf(fp, "%d %d", &isDirected, &isWeighted);
    fscanf(fp, "%d %d", &n, &m);
    if (isDirected && !isWeighted) {
        generateAdjMatrix(fp);
    }else {
        generateAdjList(fp);
    }
    fclose(fp);
    fp = fopen("fgraph.txt", "r");
    fscanf(fp, "%d %d", &isDirected, &isWeighted);
    fscanf(fp, "%d %d", &n, &m);
    if (isDirected && !isWeighted) {
        generateAdjMatrix(fp);
    } else {
        generateAdjList(fp);
    }
    fclose(fp);
    printf("\nDFS: ");
    int visited[MAX_NODES] = {0};
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i, visited);
        }
    }
    printf("\nBFS: ");
    bfs(0);
    printf("\n");

    return 0;
}
```

Output



directed weighted graph



weighted undirected graph