Dharmsinh Desai University

Academic Year 2022-23

Department:

Faculty of Management and information science

Subject: Data Structures

Full Name: Valaki Jaymin Dhirubhai

Roll No.: MA075

ID No.: 22MAPOG014

Submitted to: Prof. Himanshu K Purohit

Student sign. **Professor sign.**

HEADER FILE -> linklist_term.h

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   char data;
   struct node *next;
} *head = NULL, *temp, *newnode;

void create(struct node **head, int data) {
   struct node *newnode = (struct node *) malloc(size of(struct node));
   if (newnode == NULL) {
      printf("No memory\n");
      exit(0);
   }
   newnode->data = data;
   newnode->next = NULL;
   *head = newnode;
}
void insertEnd(struct node **head, int data) {
   struct node *temp, *newnode = (struct node *)malloc(sizeof(struct node));
   if (newnode == NULL) {
      printf("No memory\n");
      exit(0);
   }
   newnode->data = data;
   newnode->next = NULL;
```

```
    if (*head == NULL) {
        *head = newnode;
        return;
    }
    temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newnode;
}
void display(struct node *head) {
    struct node *temp;
    if (head == NULL) {
        printf("List is empty\n");
    }
    else {
        temp = head;
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}
void insertFirst(struct node **head, int data) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
```

```
        printf("No memory\n");
        exit(0);
    }
    newnode->data = data;
    newnode->next = NULL;
    if (*head == NULL) {
        *head = newnode;
        return;
    }
    newnode->next = *head;
    *head = newnode;
}

void insertSpecific(struct node **head, int data, int pos) {
    struct node *temp, *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("No memory\n");
        exit(0);
    }
    newnode->data = data;
    newnode->next = NULL;

    if (pos == 1) {
        insertFirst(head, data);
        return;
    }
```

```
    temp = *head;

    for(int i=1; i<pos-1 && temp!=NULL; i++) {
       temp=temp->next;
    }

    if(temp==NULL){
       printf("Invalid position\n");
       return;
    }

    newnode->next=temp->next;
    temp->next=newnode;
}

void deletefirst(struct node **head)
{
  struct node* temp=*head;
  if(*head==NULL)
  {
     printf("list is empty\n");
     return;
  }
  else{
     *head=temp->next;
     free(temp);
  }
```

```
}

void deleteend(struct node **head)
{
  struct node* previous=NULL;
  struct node* temp=*head;

  if(*head==NULL)
  {
     printf("list is empty\n");
     return;
  }

  else{

   while(temp->next!=NULL)
   {
     previous=temp;
     temp=temp->next;
   }

   previous->next=NULL;

   free(temp);

  }
```

```
}
void deletespecific(struct node **head, int pos)
{
   struct node *temp=*head;
   struct node *prev=NULL;

   if(*head==NULL)
   {
      printf("List is empty\n");
      return;
   }

   if(pos==1)
   {
      deletefirst(head);
      return;
   }

   for(int i=1; i<pos && temp!=NULL; i++)
   {
      prev=temp;
      temp=temp->next;
   }

   if(temp==NULL)
   {
      printf("Invalid position\n");
```

```
        return;
    }

    prev->next=temp->next;

    free(temp);
}

void reverse(struct node **head) {
    struct node *prev = NULL;
    struct node *next = NULL;
    struct node *temp = *head;
    while(temp != NULL) {
        next = temp->next;
        temp->next = prev;
        prev = temp;
        temp = next;
    }
    *head = prev;
}
```

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 :

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

struct node {
   int cust_id;
   char name[20];
   char phone_number[10];
   struct node *next;
};

//create a node
void create(struct node **newnode, int cust_id, char name[], char phone_number[]) {
   *newnode = (struct node*)malloc(sizeof(struct node));

   if(*newnode == NULL) {
      printf("No memory\n");
      exit(0);
   }

   (*newnode)->cust_id = cust_id;
   strcpy((*newnode)->name, name);
   strcpy((*newnode)->phone_number, phone_number);
   (*newnode)->next = NULL;
}

void insertend(struct node **head, int cust_id, char name[], char phone_number[]) {
   struct node *newnode;
   create(&newnode, cust_id, name, phone_number);

   if(*head == NULL) {
      *head = newnode;
      newnode->next = *head;
   } else {
      struct node *temp = *head;
      while(temp->next != *head) {
         temp = temp->next;
      }
      temp->next = newnode;
```

```
        newnode->next = *head;
    }
}

void insertfirst(struct node **head, int cust_id, char name[], char phone_number[]) {
    struct node *newnode;
    create(&newnode, cust_id, name, phone_number);

    if(*head == NULL) {
        *head = newnode;
        newnode->next = *head;
    } else {
        struct node *temp = *head;
        while(temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newnode;
        newnode->next = *head;
        *head = newnode;
    }
}

void Display(struct node *head) {
    struct node *temp = head;

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

int main()
{
    struct node *head = NULL;
    int ch;
    int cust_id;
    char name[20], phone_number[10];

    printf("Enter your choice");
    scanf("%d", &ch);

    printf("Enter customer ID, name, and phone number: ");
    scanf("%d %s %s", &cust_id, name, phone_number);

    switch (ch) {
        case 1:
```

```
            insertfirst(&head, cust_id, name, phone_number);
            break;
        case 2:
            insertend(&head, cust_id, name, phone_number);
            break;
        default:
            printf("Invalid choice\n");
            break;
    }
    return 0;
}
```

OUTPUT:

```
PS D:\VS-CODE\DATA_STRUCTURE_C> cd "d:\VS-CODE\DATA_STRUCTURE_C\" ; if ($?) { gcc T1.c -o T1 } ; if ($?) { .\T1 }
Enter the elements of the linked list:
1 2 3 4 5 -1
Enter the number of nodes to append: 3
1 2 3 4 5
3 4 5 1 2
PS D:\VS-CODE\DATA_STRUCTURE_C> cd "d:\VS-CODE\DATA_STRUCTURE_C\" ; if ($?) { gcc T1.c -o T1 } ; if ($?) { .\T1 }
Enter the elements of the linked list:
1 2 3 4 5
-1
Enter the number of nodes to append: 4
1 2 3 4 5
2 3 4 5 1
PS D:\VS-CODE\DATA_STRUCTURE_C>
```

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

Code:

A

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

struct node {
   char item;
   struct node *next;
};

struct node *temp, *first = NULL, *newnd, *last = NULL;

void create(char data) {
   newnd = (struct node*) malloc(sizeof(struct node));
   if(newnd == NULL) {
      printf("\nMemory not allocated!!");
      return;
   }
   newnd->item = data;
   newnd->next = NULL;
}

void push(char data) {
   create(data);
   if(first == NULL) {
      first = last = newnd;
   }
   else {
      newnd->next = first;
      first = newnd;
   }
}

char peek() {
   if(first == NULL) {
      printf("\nstack not exist!!");
      return -1;
```

```
   }
   else {
      return first->item;
   }
}
char pop() {
   if(first == NULL) {
      printf("\nstack not exist!!");
      return -1;
   }
   temp = first;
   char tmp = temp->item;
   first = first->next;
   free(temp);
   return tmp;
}
void display() {
   if(first == NULL) {
      printf("\nStack not exist!!");
      return;
   }
   else {
      temp = first;
      printf("\nStack:");
      while(temp != NULL) {
         printf("\n|%c|",temp->item);
         temp = temp->next;
      }
   }
}
int priority(char op) {
   if(op == '(')
      return 0;
   else if(op == '+' || op == '-')
      return 1;
   else if(op == '/' || op == '*')
      return 2;
}
int main() {
   char infix[100], N;
   int cnt = 0, i;

   printf("\nEnter Infix Expression:");
   for (i = 0; i < 100; i++) {
      scanf("%c", &infix[i]);
```

```
      if (infix[i] == '\n')
         break;
      else if(infix[i] == 32)
         continue;
      cnt++;
   }
   infix[i] = '\0';
   printf("\nPostfix Expression:");
   i = 0;
   while(infix[i] != '\0') {
      if(isalnum(infix[i])) {
         printf("%c",infix[i]);
      }
      else if (infix[i] == '(') {
         push(infix[i]);
      }
      else if(infix[i] == ')') {
         while((N = pop()) != '(') {
            printf("%c",N);
         }
      }
      else {
         while(first != NULL && priority(peek()) >= priority(infix[i])) {
            printf("%c",pop());
         }
         push(infix[i]);
 }
      i++;
   }
   while(first != NULL) {
      printf("%c",pop());
   }
   return 0;
}
```

```
PS D:\VS-CODE\DATA_STRUCTURE_C> cd "d:\VS-CODE\DATA_STRUCTURE_C\" ; if ($?) { gcc T2.c -o T2 } ; if ($?) { .\T2 }

Enter Infix Expression:a+b-c*d/e

Postfix Expression:ab+cd*e/-
PS D:\VS-CODE\DATA_STRUCTURE_C>
```

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:

A

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "linklist_term.h"

// function to find the intersection and union of two linked lists

//head 1 for linklist 1 , head2 for linklist 2, head3 for intersection, head4 for union
void interunion(struct node **head1, struct node **head2, struct node **head3, struct node **head4)
{
  // checks if both lists are empty
  if (*head1 == NULL && *head2 == NULL)
  {
    return;
  }

  struct node *temp1 = *head1;

  while (temp1 != NULL)
  {
    struct node *temp2 = *head2;
    while (temp2 != NULL)
    {
  //it compares each element of the first list with each element of the second list.
  //If an element is found in both lists, it is added to the intersection list (head3).

      if (temp1->data == temp2->data)
      {
        insertEnd(head3, temp1->data);
        break;
      }
      temp2 = temp2->next;
    }

    //otherwise add to Union list(head4)
```

```
        insertEnd(head4, temp1->data);
        temp1 = temp1->next;
    }

    // second list element that were not already added to the union list are also added to it.

    struct node *temp2 = *head2;
    while (temp2 != NULL)
    {
        struct  node *temp3 = *head3;
        bool found = false;
        while (temp3 != NULL)
        {
            //if the element was already added to the intersection list, it is not added to the union list.
            if (temp2->data == temp3->data)
            {
                found = true;
                break;
            }
            temp3 = temp3->next;
        }
        //if the element was not found in the intersection list, it is added to the union list.
        if (!found)
        {
            insertEnd(head4, temp2->data);
        }
        temp2 = temp2->next;
    }
}


int main() {
    struct node *head1 = NULL, *head2 = NULL, *head3 = NULL, *head4 = NULL;
    int data;
    printf("Enter the elements of the linked list 1:\n");
    while (1) {
        scanf("%d", &data);
        if (data == -1)
            break;
        insertEnd(&head1, data);
    }
    printf("Enter the elements of the linked list 2:\n");
    while (1) {
        scanf("%d", &data);
        if (data == -1)
```

```
            break;
        insertEnd(&head2, data);
    }
    interunion(&head1, &head2, &head3, &head4);
    printf("Intersection List: ");
    display(head3);
    printf("Union List: ");
    display(head4);
    return 0;
}
```

OUTPUT:

```
PS D:\VS-CODE\DATA_STRUCTURE_C> cd "d:\VS-CODE\DATA_STRUCTURE_C\" ; if ($?) { gcc T3.c -o T3 } ; if ($?) { .\T3 }
Enter the elements of the linked list 1:
1 2 3 4 5
-1
Enter the elements of the linked list 2:
2 3 6 7 -1
Intersection List: 2 3
Union List: 1 2 3 4 5 6 7
PS D:\VS-CODE\DATA_STRUCTURE_C>
```

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**

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

```
#include<string.h>
typedef struct mylist{
   int cust_id;
   char name[20];
   char phone_number[13];
   struct mylist *next;
}mylist;
typedef struct dial{
   mylist *head;
}dial;
mylist *createmylist(int cust_id,char name[],char phone_number[]){
   mylist *new = (mylist *)malloc(sizeof(mylist));
   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(dial *d,int cust_id){
   if(d->head == NULL){
      return 0;
   }
   mylist *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(dial *d,int cust_id,char name[],char phone_number[]){
   if(exists(d,cust_id)){
      printf("Already exists");
      return;
   }
   mylist *new = createmylist(cust_id,name,phone_number);
   if(d->head == NULL){
      d->head = new;
      new->next = new;
```

```
        return;
    }
    mylist *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(dial *d,int cust_id,char name[],char phone_number[]){
    if(exists(d,cust_id)){
        printf("Already exists");
        return;
    }
    mylist *new = createmylist(cust_id,name,phone_number);
    if(d->head == NULL){
        d->head = new;
        new->next = new;
        return;
    }
    mylist *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(dial *d,int cust_id,char name[],char phone_number[]){

    mylist *new = createmylist(cust_id,name,phone_number);
    if(d->head == NULL){
        d->head = new;
        new->next = new;
        return;
    }
    mylist *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(dial *d){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    mylist *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(dial *d){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    mylist *tmp = d->head;
    while(tmp->next->next != d->head){
        tmp = tmp->next;
    }
    mylist *tmp2 = tmp->next;
    tmp->next = d->head;
    free(tmp2);
}
// delete from specific
void deleteFromSpecific(dial *d,int cust_id){
    if(d->head == NULL){
        printf("Empty");
        return;  }
    mylist *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;
    }
    mylist *tmp2 = tmp->next;
    tmp->next = tmp->next->next;
    free(tmp2);
}

// Search by name ----------------------------------------
void searchByName(dial *d,char name[]){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    mylist *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");
}
void searchByCustId(dial *d,int cust_id){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    mylist *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(dial *d,char phone_number[]){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    mylist *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(dial *d,char name[]){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    mylist *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;
    }
    mylist *tmp2 = tmp->next;
    tmp->next = tmp->next->next;
    free(tmp2);
}
// delete by cust_id
void deleteByCustId(dial *d,int cust_id){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    mylist *tmp = d->head;
    if(tmp->cust_id == cust_id){
        while(tmp->next != d->head){
```

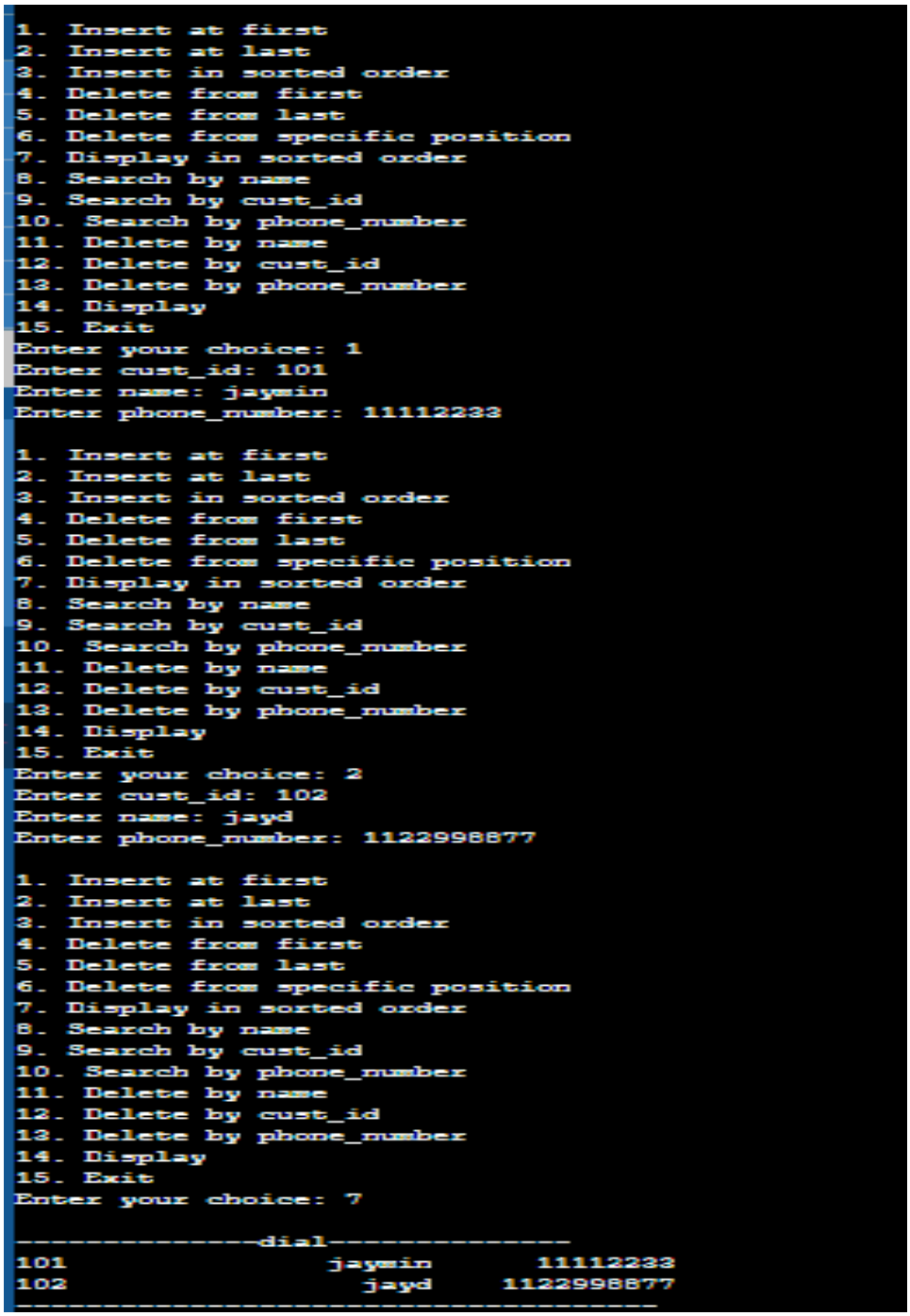
```
            tmp = tmp->next;
        }
        tmp->next = d->head->next;
        tmp = d->head;
        d->head = d->head->next;
        free(tmp);
        return;
    }
    while(tmp->next != d->head && tmp->next->cust_id != cust_id){
        tmp = tmp->next;
    }
    if(tmp->next == d->head){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

```
    free(tmp2);
}
// display
void display(dial *d){
    if(d->head == NULL){
        printf("Empty");
        return;
    }
    printf("\n--------------dial--------------");
    mylist *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(dial *d){
    // copy the list then sort that
    dial d1;
    d1.head = NULL;
    mylist *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
    dial 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 dial 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;
}
```

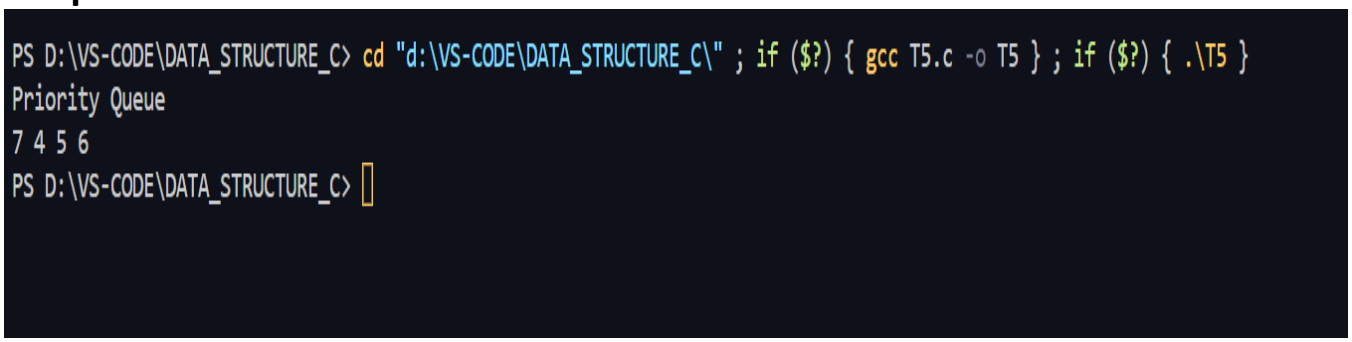

1. Insert at first
2. Insert at last
3. Insert in sorted order
4. Delete from first
5. Delete from last
6. Delete from specific position
7. Display in sorted order
8. Search by name
9. Search by cust_id
10. Search by phone_number
11. Delete by name
12. Delete by cust_id
13. Delete by phone_number
14. Display
15. Exit
Enter your choice: 1
Enter cust_id: 101
Enter name: jaymin
Enter phone_number: 11112233
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: 2
Enter cust_id: 102
Enter name: jayd
Enter phone_number: 1122998877
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: 7
--------------dial--------------
101 jaymin 11112233
102 jayd 1122998877

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

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   int priority;
   struct node *next;
} *newnode, *temp, *head;
void create(int data, int priority)
{
   newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = data;
   newnode->priority = priority;
   newnode->next = NULL;
}
int peek()
{
   return head->data;
}
void dequeue()
{
   temp = head;
   head = head->next;
   free(temp);
}
void enqueue(int data, int priority)
{
   struct node *start = head;
   create(data, priority);
   if (head->priority > priority)
   {
      newnode->next = head;
      head = newnode;
   }
   else
   {
      while (start->next != NULL && start->next->priority < priority)
      {
         start = start->next;
      }
      newnode->next = start->next;
      start->next = newnode;
   }
```

```
}
int isEmpty()
{
   return head == NULL;
}
int main()
{
   head = (struct node *)malloc(sizeof(struct node));
   head->data = 4;
   head->priority = 1;
   head->next = NULL;
   enqueue(5, 2);
   enqueue(6, 3);
   enqueue(7, 0);
   while (!isEmpty())
   {
      printf("%d ", peek());
      dequeue();
   }
   return 0;
}
```

Output



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

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;
}

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 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;
}
```

```
PS D:\VS-CODE\DATA_STRUCTURE_C> cd "d:\VS-CODE\DATA_STRUCTURE_C\" ; if ($?) { gcc t61.c -o t61 } ; if ($?) { .\t61 }

Adjacency Matrix:
0 1 1 0 0 0 1 1
0 0 0 1 1 1 1 0
0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
1 1 0 0 0 0 0 1
0 0 0 0 1 1 0 0

Adjacency List:
0: 1 2 3 8
1: 0 2 3
2: 0 1 3
3: 0 1 2
4: 8 9
5: 10
6: 7 8
7: 6
8: 0 4 6 9 11
9: 4 8 10
10: 5 9 12
11: 8
12: 10

DFS: 0 1 3 4 5 7 6 2 8 9 10 11 12
11: 8
12: 10

DFS: 0 1 3 4 5 7 6 2 8 9 10 11 12
BFS: 0 1 2 6 7 3 4 5
PS D:\VS-CODE\DATA_STRUCTURE_C>
```