ffff

**Dharmsinh Desai University**



**Academic Year: 2022-23 Department: Faculty of Management &**

**Information Science**

**Subject: Data Structure TermWork**

**Submited To :**

**Full Name: Alyani Mamad Prof. Himanshu Purohit**

**Roll No: MA003 MCA**

**ID No : 22MAPBG029 Department**

**Student Sign: Professor Sign:**

**Question – 1 :**

Write a program to compare two strings stored in singly linked list.

[ Given two strings, represented as linked lists (every character is a node in a linked list). Write

a function compare () that works similar to strcmp(), i.e., it returns 0 if both strings are the

same, 1 if the first linked list is lexicographically greater, and -1 if the second string is

lexicographically greater.]

**Source Code :**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct node {

char data;

struct node\* next;

} Node;

Node\* newNode(char data) {

Node\* node = (Node\*)malloc(sizeof(Node));

node->data = data;

node->next = NULL;

return node;

}

Node\* createLinkedList(char\* str) {

Node\* head = NULL, \* prev = NULL;

for (int i = 0; i < strlen(str); i++) {

Node\* node = newNode(str[i]);

if (prev) {

prev->next = node;

}

else {

head = node;

}

prev = node;

}

return head;

}

int compare(Node\* list1, Node\* list2) {

while (list1 && list2 && list1->data == list2->data) {

list1 = list1->next;

list2 = list2->next;

}

if (list1 && list2) {

return (list1->data > list2->data) ? 1 : -1;

}

else if (list1) {

return 1;

}

else if (list2) {

return -1;

}

else {

return 0;

}

}

int main() {

char str1[100], str2[100];

printf("Enter the first string: ");

scanf("%s", str1);

printf("Enter the second string: ");

scanf("%s", str2);

Node\* list1 = createLinkedList(str1);

Node\* list2 = createLinkedList(str2);

int result = compare(list1, list2);

printf("\n");

printf("========================================\n");

printf(" String Comparison \n");

printf("========================================\n\n");

printf("String 1: %s\n", str1);

printf("String 2: %s\n", str2);

printf("\n");

if (result == 0) {

printf("The two strings are the same.\n");

}

else if (result == 1) {

printf("The first string is lexicographically greater.\n");

}

else {

printf("The second string is lexicographically greater.\n");

}

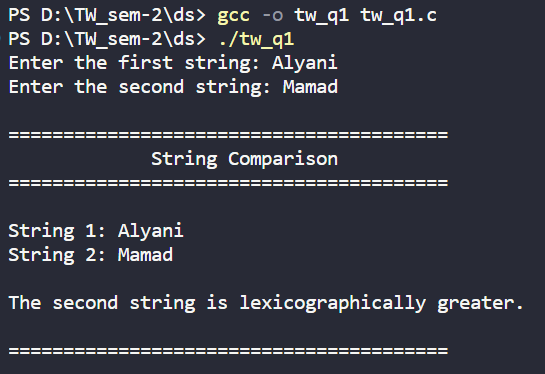
printf("\n");

printf("========================================\n");

printf("\n");

return 0;

}

**Output :**

**Question – 2 :**  Write a program to implement stack using linked list which converts infix to prefix.

**Source Code :**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct node

{

char data;

struct node \*next;

}

\*top = NULL,

\*pstart = NULL;

void insert(char ch)

{

struct node \*t, \*ba;

ba = (struct node \*)malloc(sizeof(struct node));

ba->next = NULL;

ba->data = ch;

t = pstart;

if (pstart == NULL)

{

pstart = ba;

}

else

{

while (t->next != NULL)

t = t->next;

t->next = ba;

}

}

void push(char symbol)

{

struct node \*p;

p = (struct node \*)malloc(sizeof(struct node));

p->data = symbol;

if (top == NULL)

{

top = p;

p->next = NULL;

}

else

{

p->next = top;

top = p;

}

}

char pop()

{

struct node \*x, \*y;

char k;

if (top == NULL)

{

printf("stack underflow\n");

return 0;

}

else

{

x = top;

top = top->next;

k = x->data;

free(x);

x = NULL;

return k;

}

}

void displaypost()

{

struct node \*to;

if (pstart == NULL)

printf(" ");

else

{

to = pstart;

while (to != NULL)

{

printf("%c", to->data);

to = to->next;

}

}

}

int precedence(char ch)

{

if (ch == '^')

return (5);

else if (ch == '\*' || ch == '/')

return (4);

else if (ch == '+' || ch == '-')

return (3);

else

return (2);

}

void intopost(char infix[])

{

int len;

int index = 0;

char symbol, temp;

len = strlen(infix);

while (len > index)

{

symbol = infix[index];

switch (symbol)

{

case '(':

push(symbol);

break;

case ')':

temp = pop();

while (temp != '(')

{

break;

}

insert(temp);

temp = pop();

case '^':

case '+':

case '-':

case '\*':

case '/':

if (top == NULL)

{

push(symbol);

}

else

{

while (top != NULL && (precedence(top -> data) >= precedence(symbol)))

{

temp = pop();

insert(temp);

}

push(symbol);

}

break;

default:

insert(symbol);

}

index = index + 1;

}

while (top != NULL)

{

temp = pop();

insert(temp);

}

displaypost();

return;

}

int main()

{

char infix[50];

printf("enter infix expression: ");

gets(infix);

printf("\n\npostfix expression is---> ");

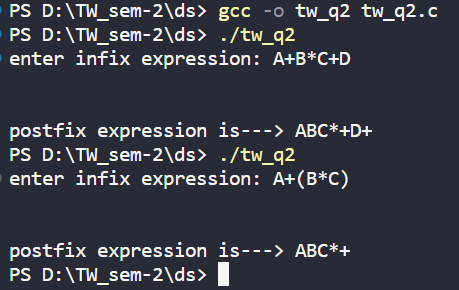
intopost(infix);

getchar();

return 0;

}

**Output :**

****

**Question – 3 :**

Write a Program to Reverse a Linked List in groups of given size

Given a linked list, write a function to reverse every k node (where k is an input to the

function).

Example:

Input: 1->2->3->4->5->6->7->8->NULL, K = 3

Output: 3->2->1->6->5->4->8->7->NULL

Input: 1->2->3->4->5->6->7->8->NULL, K = 5

Output: 5->4->3->2->1->8->7->6->NULL ]

**Source Code :**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* reverseGroup(struct Node\* head, int k) {

struct Node\* current = head;

struct Node\* next = NULL;

struct Node\* prev = NULL;

int count = 0;

while (current != NULL && count < k) {

next = current->next;

current->next = prev;

prev = current;

current = next;

count++;

}

if (next != NULL) {

head->next = reverseGroup(next, k);

}

return prev;

}

void printList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

printf("%d->", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

void insert(struct Node\*\* head, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

int main() {

int k = 0;

printf("\nEnter Value Of K : ");

scanf("%d",&k);

struct Node\* head = NULL;

insert(&head, 8);

insert(&head, 7);

insert(&head, 6);

insert(&head, 5);

insert(&head, 4);

insert(&head, 3);

insert(&head, 2);

insert(&head, 1);

printf("\nOriginal Linked List: ");

printList(head);

head = reverseGroup(head, k);

printf("K = %d\n",k);

printf("Reversed Linked List: ");

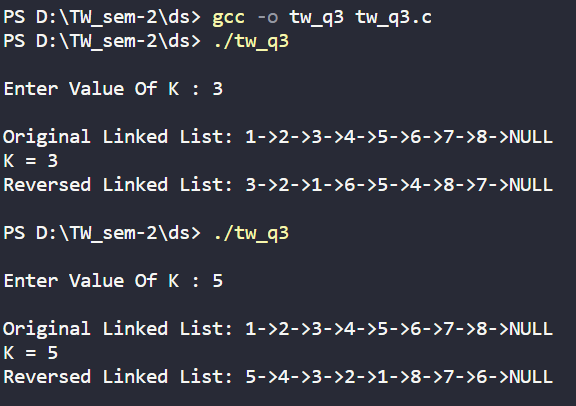
printList(head);

printf("\n");

return 0;

}

**Output :**

****

**Question – 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

**Source Code :**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX\_NAME 100

#define MAX\_PHONE\_NUMBER 100

typedef struct node {

int cust\_id;

char name[MAX\_NAME];

char phone\_number[MAX\_PHONE\_NUMBER];

struct node\* next;

}Node;

Node\* head = NULL;

Node\* tail = NULL;

void insertFromFirst(int cust\_id, char name[], char phone\_number[]) {

Node\* newNode = ( Node\*) malloc(sizeof( Node));

newNode->cust\_id = cust\_id;

strcpy(newNode->name, name);

strcpy(newNode->phone\_number, phone\_number);

if (head == NULL) {

head = newNode;

tail = newNode;

newNode->next = head;

} else {

newNode->next = head;

head = newNode;

tail->next = head;

}

printf("\nNode with customer ID %d inserted at the beginning.", cust\_id);

}

void insertFromLast(int cust\_id, char name[], char phone\_number[]) {

Node\* newNode = ( Node\*) malloc(sizeof( Node));

newNode->cust\_id = cust\_id;

strcpy(newNode->name, name);

strcpy(newNode->phone\_number, phone\_number);

if (head == NULL) {

head = newNode;

tail = newNode;

newNode->next = head;

} else {

tail->next = newNode;

tail = newNode;

tail->next = head;

}

printf("\nNode with customer ID %d inserted at the end.", cust\_id);

}

void insertAtPosition(int position, int cust\_id, char name[], char phone\_number[]) {

if (head == NULL) {

printf("\nList is empty. Cannot insert at position %d.", position);

return;

}

int i;

Node\* temp = head;

Node\* prev = NULL;

for (i = 0; i < position-1; i++) {

prev = temp;

temp = temp->next;

if (temp == head && i != position-2) {

printf("\nInvalid position. Cannot insert at position %d.", position);

return;

}

}

Node\* newNode = ( Node\*) malloc(sizeof( Node));

newNode->cust\_id = cust\_id;

strcpy(newNode->name, name);

strcpy(newNode->phone\_number, phone\_number);

if (prev == NULL) {

newNode->next = head;

head = newNode;

tail->next = head;

} else if (temp == head && i == position-1) {

tail->next = newNode;

newNode->next = head;

tail = newNode;

} else {

prev->next = newNode;

newNode->next = temp;

}

printf("\nNode with customer ID %d inserted at position %d.", cust\_id, position);

}

void deleteFromPosition(int position) {

if (head == NULL) {

printf("\nList is empty. Cannot delete from position %d.", position);

return;

}

int i;

Node\* temp = head;

Node\* prev = NULL;

for (i = 0; i < position-1; i++) {

prev = temp;

temp = temp->next;

if (temp == head && i!= position-2) {

printf("\nInvalid position. Cannot delete from position %d.", position);

return;

}

}

if (prev == NULL) {

head = temp->next;

tail->next = head;

free(temp);

} else if (temp == tail) {

prev->next = head;

tail = prev;

free(temp);

} else {

prev->next = temp->next;

free(temp);

}

printf("\nNode deleted from position %d.", position);

}

void deleteFromFirst() {

if (head == NULL) {

printf("\nList is empty. Cannot delete from first.");

return;

}

Node\* temp = head;

if (head == tail) {

head = NULL;

tail = NULL;

} else {

head = head->next;

tail->next = head;

}

free(temp);

printf("\nNode deleted from the beginning.");

}

void deleteFromLast() {

if (head == NULL) {

printf("\nList is empty. Cannot delete from last.");

return;

}

Node\* temp = head;

Node\* prev = NULL;

while (temp->next != head) {

prev = temp;

temp = temp->next;

}

if (prev == NULL) {

head = NULL;

tail = NULL;

} else {

prev->next = head;

tail = prev;

}

free(temp);

printf("\nNode deleted from the end.");

}

void displaySorted() {

if (head == NULL) {

printf("\nList is empty. Cannot display sorted.");

return;

}

Node\* current = head;

Node\* index = NULL;

int tempCustId;

char tempName[50];

char tempPhoneNumber[20];

if (head == tail) {

printf("\nCustomer ID: %d, Name: %s, Phone Number: %s", head->cust\_id, head->name, head->phone\_number);

} else {

do {

index = current->next;

while (index != head) {

if (strcmp(current->name, index->name) > 0) {

tempCustId = current->cust\_id;

strcpy(tempName, current->name);

strcpy(tempPhoneNumber, current->phone\_number);

current->cust\_id = index->cust\_id;

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

strcpy(current->phone\_number, index->phone\_number);

index->cust\_id = tempCustId;

strcpy(index->name, tempName);

strcpy(index->phone\_number, tempPhoneNumber);

}

index = index->next;

}

printf("\nCustomer ID: %d, Name: %s, Phone Number: %s", current->cust\_id, current->name, current->phone\_number);

current = current->next;

} while (current != head);

}

}

void searchByName(char name[]) {

if (head == NULL) {

printf("\nList is empty. Cannot search by name.");

return;

}

Node\* current = head;

int found = 0;

do {

if (strcmp(current->name, name) == 0) {

printf("\nCustomer ID: %d, Name: %s, Phone Number: %s", current->cust\_id, current->name, current->phone\_number);

found = 1;

}

current = current->next;

} while (current != head);

if (!found) {

printf("\nCustomer with name %s not found.", name);

}

}

void searchByCustId(int cust\_id) {

if (head == NULL) {

printf("\nList is empty. Cannot search by customer ID.");

return;

}

Node\* current = head;

int found = 0;

do {

if (current->cust\_id == cust\_id) {

printf("\nCustomer ID: %d, Name: %s, Phone Number: %s", current->cust\_id, current->name, current->phone\_number);

found = 1;

}

current = current->next;

} while (current != head);

if (!found) {

printf("\nCustomer with ID %d not found.", cust\_id);

}

}

void searchByPhoneNumber(char\* phone\_number) {

if (head == NULL) {

printf("\nList is empty. Cannot search by phone number.");

return;

}

Node\* current = head;

int found = 0;

do {

if (strcmp(current->phone\_number, phone\_number) == 0) {

printf("\nCustomer ID: %d, Name: %s, Phone Number: %s", current->cust\_id, current->name, current->phone\_number);

found = 1;

}

current = current->next;

} while (current != head);

if (!found) {

printf("\nCustomer with phone number %s not found.", phone\_number);

}

}

void deleteByName(char name[]) {

if (head == NULL) {

printf("\nList is empty. Cannot delete by name.");

return;

}

Node\* temp = head;

Node\* prev = NULL;

int found = 0;

do {

if (strcmp(temp->name, name) == 0) {

found = 1;

break;

}

prev = temp;

temp = temp->next;

} while (temp != head);

if (found) {

if (prev == NULL) {

head = temp->next;

tail->next = head;

free(temp);

} else if (temp == tail) {

prev->next = head;

tail = prev;

free(temp);

} else {

prev->next = temp->next;

free(temp);

}

printf("\nNode deleted with name %s.", name);

} else {

printf("\nCustomer with name %s not found.", name);

}

}

void deleteByCustId(int cust\_id) {

if (head == NULL) {

printf("\nList is empty. Cannot delete by customer ID.");

return;

}

Node\* temp = head;

Node\* prev = NULL;

int found = 0;

do {

if (temp->cust\_id == cust\_id) {

found = 1;

break;

}

prev = temp;

temp = temp->next;

} while (temp != head);

if (found) {

if (prev == NULL) {

head = temp->next;

tail->next = head;

free(temp);

} else if(temp == tail) {

prev->next = head;

tail = prev;

free(temp);

} else {

prev->next = temp->next;

free(temp);

}

printf("\nNode deleted with customer ID %d.", cust\_id);

} else {

printf("\nCustomer with customer ID %d not found.", cust\_id);

}

}

void deleteByPhoneNumber(char phone\_number[]) {

if (head == NULL) {

printf("\nList is empty. Cannot delete by phone number.");

return;

}

Node\* temp = head;

Node\* prev = NULL;

int found = 0;

do {

if (strcmp(temp->phone\_number, phone\_number) == 0) {

found = 1;

break;

}

prev = temp;

temp = temp->next;

} while (temp != head);

if (found) {

if (prev == NULL) {

head = temp->next;

tail->next = head;

free(temp);

} else if (temp == tail) {

prev->next = head;

tail = prev;

free(temp);

} else {

prev->next = temp->next;

free(temp);

}

printf("\nNode deleted with phone number %s.", phone\_number);

} else {

printf("\nCustomer with phone number %s not found.", phone\_number);

}

}

int main() {

int choice, cust\_id, pos;

char name[MAX\_NAME];

char phone\_number[MAX\_PHONE\_NUMBER];

while (1) {

printf("\n------------------------------------------------------\n");

printf("PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST\n");

printf("------------------------------------------------------\n");

printf("1. Insert from first\n");

printf("2. Insert from last\n");

printf("3. Insert at specific position\n");

printf("4. Delete from specific position\n");

printf("5. Delete from first\n");

printf("6. Delete from last\n");

printf("7. Display in sorted order\n");

printf("8. Search by name\n");

printf("9. Search by customer ID\n");

printf("10. Search by phone number\n");

printf("11. Delete by name\n");

printf("12. Delete by customer ID\n");

printf("13. Delete by phone number\n");

printf("14. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\nEnter customer ID: ");

scanf("%d", &cust\_id);

printf("Enter name: ");

scanf("%s", name);

printf("Enter phone number: ");

scanf("%s", phone\_number);

insertFromFirst(cust\_id, name, phone\_number);

break;

case 2:

printf("\nEnter customer ID: ");

scanf("%d", &cust\_id);

printf("Enter name: ");

scanf("%s", name);

printf("Enter phone number: ");

scanf("%s", phone\_number);

insertFromLast(cust\_id, name, phone\_number);

break;

case 3:

printf("\nEnter customer ID: ");

scanf("%d", &cust\_id);

printf("Enter name: ");

scanf("%s", name);

printf("Enter phone number: ");

scanf("%s", phone\_number);

printf("Enter position: ");

scanf("%d", &pos);

insertAtPosition(pos,cust\_id, name, phone\_number);

break;

case 4:

printf("\nEnter position: ");

scanf("%d", &pos);

deleteFromPosition(pos);

break;

case 5:

deleteFromFirst();

break;

case 6:

deleteFromLast();

break;

case 7:

displaySorted();

break;

case 8:

printf("\nEnter name to search: ");

scanf("%s", name);

searchByName(name);

break;

case 9:

printf("\nEnter customer ID to search: ");

scanf("%d", &cust\_id);

searchByCustId(cust\_id);

break;

case 10:

printf("\nEnter phone number to search: ");

scanf("%s", phone\_number);

searchByPhoneNumber(phone\_number);

break;

case 11:

printf("\nEnter name to delete: ");

scanf("%s", name);

deleteByName(name);

break;

case 12:

printf("\nEnter customer ID to delete: ");

scanf("%d", &cust\_id);

deleteByCustId(cust\_id);

break;

case 13:

printf("\nEnter phone number to delete: ");

scanf("%s", phone\_number);

deleteByPhoneNumber(phone\_number);

break;

case 14:

printf("\nExiting program...");

exit(0);

break;

default:

printf("\nInvalid choice. Please try again.");

}

}

return 0;

}

**Output :**

PS D:\TW\_sem-2\ds> gcc -o tw\_q4 tw\_q4.c

PS D:\TW\_sem-2\ds> ./tw\_q4

------------------------------------------------------

PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST

------------------------------------------------------

1. Insert from first

2. Insert from last

3. Insert at specific position

4. Delete from specific position

5. Delete from first

6. Delete from last

7. Display in sorted order

8. Search by name

9. Search by customer ID

10. Search by phone number

11. Delete by name

12. Delete by customer ID

13. Delete by phone number

14. Exit

Enter your choice: 1

Enter customer ID: 21

Enter name: alyani

Enter phone number: 7984657346

Node with customer ID 21 inserted at the beginning.

------------------------------------------------------

PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST

------------------------------------------------------

1. Insert from first

2. Insert from last

3. Insert at specific position

4. Delete from specific position

5. Delete from first

6. Delete from last

7. Display in sorted order

8. Search by name

9. Search by customer ID

10. Search by phone number

11. Delete by name

12. Delete by customer ID

13. Delete by phone number

14. Exit

Enter your choice: 2

Enter customer ID: 22

Enter name: akshay

Enter phone number: 6355948766

Node with customer ID 22 inserted at the end.

------------------------------------------------------

PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST

------------------------------------------------------

1. Insert from first

2. Insert from last

3. Insert at specific position

4. Delete from specific position

5. Delete from first

6. Delete from last

7. Display in sorted order

8. Search by name

9. Search by customer ID

10. Search by phone number

11. Delete by name

12. Delete by customer ID

13. Delete by phone number

14. Exit

Enter your choice: 3

Enter customer ID: 23

Enter name: pathu

Enter phone number: 99055547586

Enter position: 3

Node with customer ID 23 inserted at position 3.

------------------------------------------------------

PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST

------------------------------------------------------

1. Insert from first

2. Insert from last

3. Insert at specific position

4. Delete from specific position

5. Delete from first

6. Delete from last

7. Display in sorted order

8. Search by name

9. Search by customer ID

10. Search by phone number

11. Delete by name

12. Delete by customer ID

13. Delete by phone number

14. Exit

Enter your choice: 7

Customer ID: 22, Name: akshay, Phone Number: 6355948766

Customer ID: 21, Name: alyani, Phone Number: 7984657346

Customer ID: 23, Name: pathu, Phone Number: 99055547586

------------------------------------------------------

PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST

------------------------------------------------------

1. Insert from first

2. Insert from last

3. Insert at specific position

4. Delete from specific position

5. Delete from first

6. Delete from last

7. Display in sorted order

8. Search by name

9. Search by customer ID

10. Search by phone number

11. Delete by name

12. Delete by customer ID

13. Delete by phone number

14. Exit

Enter your choice: 5

Node deleted from the beginning.

------------------------------------------------------

PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST

------------------------------------------------------

1. Insert from first

2. Insert from last

3. Insert at specific position

4. Delete from specific position

5. Delete from first

6. Delete from last

7. Display in sorted order

8. Search by name

9. Search by customer ID

10. Search by phone number

11. Delete by name

12. Delete by customer ID

13. Delete by phone number

14. Exit

Enter your choice: 7

Customer ID: 21, Name: alyani, Phone Number: 7984657346

Customer ID: 23, Name: pathu, Phone Number: 99055547586

------------------------------------------------------

PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST

------------------------------------------------------

1. Insert from first

2. Insert from last

3. Insert at specific position

4. Delete from specific position

5. Delete from first

6. Delete from last

7. Display in sorted order

8. Search by name

9. Search by customer ID

10. Search by phone number

11. Delete by name

12. Delete by customer ID

13. Delete by phone number

14. Exit

Enter your choice: 4

Enter position: 0

Node deleted from position 0.

------------------------------------------------------

PHONE DIRECTORY USING SINGLY CIRCULAR LINKED LIST

------------------------------------------------------

1. Insert from first

2. Insert from last

3. Insert at specific position

4. Delete from specific position

5. Delete from first

6. Delete from last

7. Display in sorted order

8. Search by name

9. Search by customer ID

10. Search by phone number

11. Delete by name

12. Delete by customer ID

13. Delete by phone number

14. Exit

Enter your choice: 14

Exiting program...

**Question – 5 :**

Write a program to implement priority queue using linked list.

**Source Code :**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

int data;

int priority;

struct Node \*next;

} Node;

Node \*create(int data, int priority)

{

Node \*node = (Node \*)malloc(sizeof(Node));

node->data = data;

node->priority = priority;

node->next = NULL;

return node;

}

void enqueue(Node \*\*head, int data, int priority)

{

Node \*newNode = create(data, priority);

if (\*head == NULL || priority > (\*head)->priority)

{

newNode->next = \*head;

\*head = newNode;

}

else

{

Node \*current = \*head;

while (current->next != NULL && current->next->priority >= priority)

{

current = current->next;

}

newNode->next = current->next;

current->next = newNode;

}

}

int dequeue(Node \*\*head)

{

if (\*head == NULL)

{

printf("Queue is empty!\n");

return -1;

}

Node \*temp = \*head;

int data = temp->data;

\*head = (\*head)->next;

free(temp);

return data;

}

void printQueue(Node \*head)

{

printf("Priority Queue: ");

while (head != NULL)

{

printf("(%d, %d) ", head->data, head->priority);

head = head->next;

}

printf("\n");

}

int main()

{

Node \*head = NULL;

int data, priority, choice;

do

{

printf("\n1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Print queue\n");

printf("0. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter data: ");

scanf("%d", &data);

printf("Enter priority: ");

scanf("%d", &priority);

enqueue(&head, data, priority);

break;

case 2:

dequeue(&head);

break;

case 3:

printQueue(head);

break;

case 0:

break;

default:

printf("Invalid choice!\n");

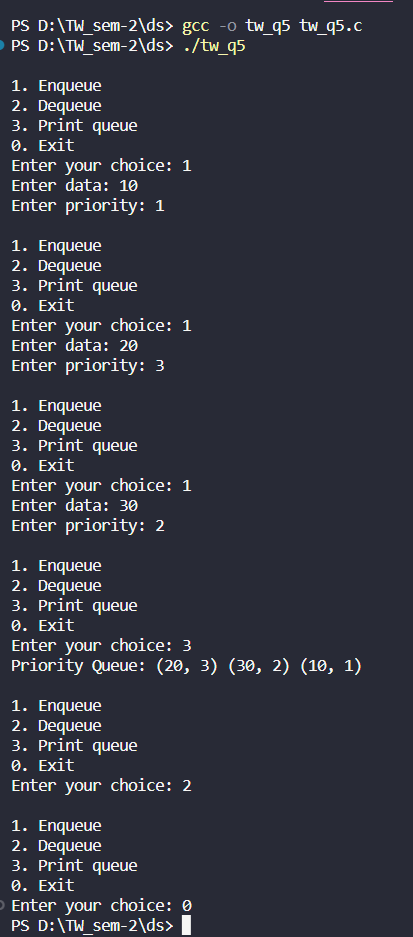
break;

}

} while (choice != 0);

return 0;

}

**Output :**

**Common-Question :**

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.

IF Directed (and Unweighted)

Generate Adjacency Matrix

Instantiate directed graph

Traverse the graph with DFS and BFS

Else Undirected

Generate Adjacency List

Instantiate undirected graph

Traverse the graph with DFS and BFS

**Source 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 printAdjList() {

printf("\nAdjacency List:\n");

for (int i = 0; i < n; i++) {

printf("%d: ", i);

for (int j = 0; j < n; j++) {

if (adjList[i][j]) {

printf("%d ", 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();

}

void generateAdjList(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);

}

adjList[u][v] = 1;

if (!isDirected) {

adjList[v][u] = 1;

}

}

printAdjList();

}

int main() {

FILE \*fp;

fp = fopen("file1.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("file2.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 :**

