#include <stdio.h>

#include <string.h>

#define MAX 100

typedef struct

{

int prn;

char name[50];

char branch[50];

} Student;

void addStudent(Student students[], int \*count);

void deleteStudent(Student students[], int \*count);

void modifyStudent(Student students[], int count);

void sortStudents(Student students[], int count);

void searchStudent(Student students[], int count);

void displayStudents(Student students[], int count);

void addStudent(Student students[], int \*count)

{

if (\*count >= MAX)

{

printf("Limit reached.\n");

return;

}

printf("PRN: ");

scanf("%d", &students[\*count].prn);

printf("Name: ");

scanf("%s", students[\*count].name);

printf("Branch: ");

scanf("%s", students[\*count].branch);

(\*count)++;

printf("Added!\n");

}

void deleteStudent(Student students[], int \*count)

{

if (\*count == 0)

{

printf("No data.\n");

return;

}

int prn, i, j;

printf("PRN to delete: ");

scanf("%d", &prn);

for (i = 0; i < \*count; i++)

{

if (students[i].prn == prn)

{

for (j = i; j < \*count - 1; j++)

{

students[j] = students[j + 1];

}

(\*count)--;

printf("Deleted!\n");

return;

}

}

printf("Not found.\n");

}

void modifyStudent(Student students[], int count)

{

if (count == 0)

{

printf("No data.\n");

return;

}

int prn, i;

printf("PRN to modify: ");

scanf("%d", &prn);

for (i = 0; i < count; i++)

{

if (students[i].prn == prn)

{

printf("New Name: ");

scanf("%s", students[i].name);

printf("New Branch: ");

scanf("%s", students[i].branch);

printf("Modified!\n");

return;

}

}

printf("Not found.\n");

}

void sortStudents(Student students[], int count)

{

if (count == 0)

{

printf("No data.\n");

return;

}

Student temp;

for (int i = 0; i < count - 1; i++)

{

for (int j = 0; j < count - i - 1; j++)

{

if (students[j].prn > students[j + 1].prn)

{

temp = students[j];

students[j] = students[j + 1];

students[j + 1] = temp;

}

}

}

printf("Sorted!\n");

displayStudents(students, count);

}

void searchStudent(Student students[], int count)

{

if (count == 0)

{

printf("No data.\n");

return;

}

int prn, i;

printf("PRN to search: ");

scanf("%d", &prn);

for (i = 0; i < count; i++)

{

if (students[i].prn == prn)

{

printf("Found: %d %s %s\n", students[i].prn, students[i].name,

students[i].branch);

return;

}

}

printf("Not found.\n");

}

void displayStudents(Student students[], int count)

{

if (count == 0)

{

printf("No data.\n");

return;

}

printf("PRN\tName\tBranch\n");

for (int i = 0; i < count; i++)

{

printf("%d\t%s\t%s\n", students[i].prn, students[i].name, students[i].branch);

}

}

int main()

{

Student students[MAX];

int count = 0;

int choice;

while (1)

{

printf("\n1.Add 2.Del 3.Mod 4.Sort 5.Search 6.Disp 7.Exit: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

addStudent(students, &count);

break;

case 2:

deleteStudent(students, &count);

break;

case 3:

modifyStudent(students, count);

break;

case 4:

sortStudents(students, count);

break;

case 5:

searchStudent(students, count);

break;

case 6:

displayStudents(students, count);

break;

case 7:

printf("Exiting.\n");

return 0;

default:

printf("Invalid choice.\n");

}

}

}

....................................................................................................

PRACTICAL 2

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

Node \*createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insertFront(Node \*\*head, int data) {

Node \*newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

printf("Inserted at front.\n");

}

void insertEnd(Node \*\*head, int data) {

Node \*newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

Node \*temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

printf("Inserted at end.\n");

}

void insertMiddle(Node \*\*head, int data, int position) {

if (position == 1) {

insertFront(head, data);

return;

}

Node \*newNode = createNode(data);

Node \*temp = \*head;

for (int i = 1; i < position - 1 && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Invalid position.\n");

return;

}

newNode->next = temp->next;

temp->next = newNode;

printf("Inserted at position %d.\n", position);

}

void deleteFront(Node \*\*head) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

Node \*temp = \*head;

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

free(temp);

printf("Deleted from front.\n");

}

void deleteEnd(Node \*\*head) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

if ((\*head)->next == NULL) {

free(\*head);

\*head = NULL;

printf("Deleted from end.\n");

return;

}

Node \*temp = \*head;

while (temp->next->next != NULL) {

temp = temp->next;

}

free(temp->next);

temp->next = NULL;

printf("Deleted from end.\n");

}

void deleteMiddle(Node \*\*head, int position) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

if (position == 1) {

deleteFront(head);

return;

}

Node \*temp = \*head;

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

temp = temp->next;

}

if (temp->next == NULL) {

printf("Invalid position.\n");

return;

}

Node \*toDelete = temp->next;

temp->next = toDelete->next;

free(toDelete);

printf("Deleted from position %d.\n", position);

}

void displayList(Node \*head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node \*temp = head;

printf("List: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

Node \*head = NULL;

int choice, data, position;

while (1) {

printf("\n1.Insert Front 2.Insert End 3.Insert Middle");

printf("\n4.Delete Front 5.Delete End 6.Delete Middle 7.Display 8.Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data: ");

scanf("%d", &data);

insertFront(&head, data);

break;

case 2:

printf("Enter data: ");

scanf("%d", &data);

insertEnd(&head, data);

break;

case 3:

printf("Enter data: ");

scanf("%d", &data);

printf("Enter position: ");

scanf("%d", &position);

insertMiddle(&head, data, position);

break;

case 4:

deleteFront(&head);

break;

case 5:

deleteEnd(&head);

break;

case 6:

printf("Enter position: ");

scanf("%d", &position);

deleteMiddle(&head, position);

break;

case 7:

displayList(head);

break;

case 8:

printf("Exiting.\n");

return 0;

default:

printf("Invalid choice.\n");

}

}

}

..................................................................................................

PRACTICAL 3

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

char data;

struct Node \*prev;

struct Node \*next;

} Node;

Node \*createNode(char data);

void constructList(Node \*\*head, const char \*str);

void insert(Node \*\*head, char data, int position);

void delete (Node \*\*head, int position);

void displayForward(Node \*head);

void displayBackward(Node \*head);

int main() {

Node \*head = NULL;

char input[100];

int choice, position;

char data;

printf("Enter a string: ");

scanf("%s", input);

constructList(&head, input);

while (1) {

printf("\n1.Insert 2.Delete 3.Display Forward 4.Display Backward 5.Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter character to insert: ");

scanf(" %c", &data);

printf("Enter position (1-based): ");

scanf("%d", &position);

insert(&head, data, position);

break;

case 2:

printf("Enter position (1-based) to delete: ");

scanf("%d", &position);

delete (&head, position);

break;

case 3:

printf("List (forward): ");

displayForward(head);

break;

case 4:

printf("List (backward): ");

displayBackward(head);

break;

case 5:

printf("Exiting.\n");

return 0;

default:

printf("Invalid choice.\n");

}

}

}

Node \*createNode(char data) {

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

// creating a doubly linked list from the input string

void constructList(Node \*\*head, const char \*str) {

Node \*temp, \*tail = NULL;

for (int i = 0; str[i] != '\0'; i++) {

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

if (\*head == NULL) {

\*head = newNode;

tail = \*head;

} else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

printf("Doubly linked list constructed.\n");

}

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

Node \*newNode = createNode(data);

if (position == 1) {

newNode->next = \*head;

if (\*head) {

(\*head)->prev = newNode;

}

\*head = newNode;

printf("Inserted at position %d.\n", position);

return;

}

Node \*temp = \*head;

for (int i = 1; i < position - 1 && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Invalid position.\n");

free(newNode);

return;

}

newNode->next = temp->next;

if (temp->next) {

temp->next->prev = newNode;

}

temp->next = newNode;

newNode->prev = temp;

printf("Inserted at position %d.\n", position);

}

void delete (Node \*\*head, int position) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

Node \*temp = \*head;

if (position == 1) {

\*head = temp->next;

if (\*head) {

(\*head)->prev = NULL;

}

free(temp);

printf("Deleted from position %d.\n", position);

return;

}

for (int i = 1; i < position && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Invalid position.\n");

return;

}

if (temp->next) {

temp->next->prev = temp->prev;

}

if (temp->prev) {

temp->prev->next = temp->next;

}

free(temp);

printf("Deleted from position %d.\n", position);

}

void displayForward(Node \*head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node \*temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

void displayBackward(Node \*head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node \*temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

while (temp != NULL) {

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

temp = temp->prev;

}

printf("\n");

}

...................................................................................................

PRACTICAL 4

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct

{

int data[MAX];

int top;

} Stack;

void initialize(Stack \*s);

int isEmpty(Stack \*s);

int isFull(Stack \*s);

void push(Stack \*s, int value);

int pop(Stack \*s);

int peek(Stack \*s);

void display(Stack \*s);

int main()

{

Stack stack;

int choice, value;

initialize(&stack);

while (1)

{

printf("\n1.Push 2.Pop 3.Peek 4.Display 5.Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

if (isFull(&stack))

{

printf("Stack is full. Cannot push.\n");

}

else

{

printf("Enter value to push: ");

scanf("%d", &value);

push(&stack, value);

}

break;

case 2:

if (isEmpty(&stack))

{

printf("Stack is empty. Cannot pop.\n");

}

else

{

value = pop(&stack);

printf("Popped value: %d\n", value);

}

break;

case 3:

if (isEmpty(&stack))

{

printf("Stack is empty.\n");

}

else

{

printf("Top element: %d\n", peek(&stack));

}

break;

case 4:

display(&stack);

break;

case 5:

printf("Exiting.\n");

exit(0);

default:

printf("Invalid choice.\n");

}

}

}

void initialize(Stack \*s)

{

s->top = -1;

}

int isEmpty(Stack \*s)

{

return s->top == -1;

}

int isFull(Stack \*s)

{

return s->top == MAX - 1;

}

void push(Stack \*s, int value)

{

if (isFull(s))

{

printf("Stack overflow. Cannot push.\n");

return;

}

s->data[++(s->top)] = value;

printf("Pushed %d onto the stack.\n", value);

}

int pop(Stack \*s)

{

if (isEmpty(s))

{

printf("Stack underflow. Cannot pop.\n");

return -1;

}

return s->data[(s->top)--];

}

int peek(Stack \*s)

{

if (isEmpty(s))

{

printf("Stack is empty.\n");

return -1;

}

return s->data[s->top];

}

void display(Stack \*s)

{

if (isEmpty(s))

{

printf("Stack is empty.\n");

return;

}

printf("Stack elements: ");

for (int i = s->top; i >= 0; i--)

{

printf("%d ", s->data[i]);

}

printf("\n");

}

......................................................................................................

PRACTICAL 5

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

int data;

struct Node \*next;

} Node;

void push(Node \*\*top, int value);

int pop(Node \*\*top);

int isEmpty(Node \*top);

void display(Node \*top);

int main()

{

Node \*stack = NULL;

int choice, value;

while (1)

{

printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(&stack, value);

break;

case 2:

if (isEmpty(stack))

{

printf("Stack is empty! Cannot pop.\n");

}

else

{

value = pop(&stack);

printf("Popped value: %d\n", value);

}

break;

case 3:

display(stack);

break;

case 4:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Try again.\n");

}

}

}

void push(Node \*\*top, int value)

{

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

if (newNode == NULL)

{

printf("Memory allocation failed! Cannot push.\n");

return;

}

newNode->data = value;

newNode->next = \*top;

\*top = newNode;

printf("Pushed %d onto the stack.\n", value);

}

int pop(Node \*\*top)

{

if (isEmpty(\*top))

{

printf("Stack is empty! Cannot pop.\n");

return -1;

}

Node \*temp = \*top;

int value = temp->data;

\*top = temp->next;

free(temp);

return value;

}

int isEmpty(Node \*top)

{

return top == NULL;

}

void display(Node \*top)

{

if (isEmpty(top))

{

printf("Stack is empty.\n");

return;

}

printf("Stack elements: ");

Node \*current = top;

while (current != NULL)

{

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

current = current->next;

}

printf("\n");

}

..............................................................................................................

PRACTICAL 6

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

typedef struct Queue

{

int front;

int rear;

int arr[MAX];

} Queue;

void enqueue(Queue \*q, int value);

int dequeue(Queue \*q);

int isEmpty(Queue \*q);

int isFull(Queue \*q);

void display(Queue \*q);

int main()

{

Queue q;

q.front = -1;

q.rear = -1;

int choice, value;

while (1)

{

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(&q, value);

break;

case 2:

if (isEmpty(&q))

{

printf("Queue is empty! Cannot dequeue.\n");

}

else

{

value = dequeue(&q);

printf("Dequeued value: %d\n", value);

}

break;

case 3:

display(&q);

break;

case 4:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Try again.\n");

}

}

}

void enqueue(Queue \*q, int value)

{

if (isFull(q))

{

printf("Queue is full! Cannot enqueue.\n");

return;

}

if (q->front == -1)

{

q->front = 0;

}

q->rear++;

q->arr[q->rear] = value;

printf("Enqueued %d into the queue.\n", value);

}

int dequeue(Queue \*q)

{

if (isEmpty(q))

{

printf("Queue is empty! Cannot dequeue.\n");

return -1;

}

int value = q->arr[q->front];

q->front++;

if (q->front > q->rear)

{

q->front = q->rear = -1;

}

return value;

}

int isEmpty(Queue \*q)

{

return q->front == -1;

}

int isFull(Queue \*q)

{

return q->rear == MAX - 1;

}

void display(Queue \*q)

{

if (isEmpty(q))

{

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

return;

}

printf("Queue elements: ");

for (int i = q->front; i <= q->rear; i++)

{

printf("%d ", q->arr[i]);

}

printf("\n");

}

.........................................................................................................................

PRACTICAL 7

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Queue {

struct Node\* front;

struct Node\* rear;

};

void initializeQueue(struct Queue\* q) {

q->front = q->rear = NULL;

}

int isEmpty(struct Queue\* q) {

return (q->front == NULL);

}

void enqueue(struct Queue\* q, int value) {

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

if (!newNode) {

printf("Memory allocation failed\n");

return;

}

newNode->data = value;

newNode->next = NULL;

if (q->rear == NULL) {

q->front = q->rear = newNode;

} else {

q->rear->next = newNode;

q->rear = newNode;

}

}

int dequeue(struct Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return -1;

}

struct Node\* temp = q->front;

int value = temp->data;

q->front = q->front->next;

if (q->front == NULL) {

q->rear = NULL;

}

free(temp);

return value;

}

int peek(struct Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return -1;

}

return q->front->data;

}

void display(struct Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

struct Node\* temp = q->front;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

struct Queue q;

initializeQueue(&q);

int choice, value;

while (1) {

printf("\n1. Enqueue\n2. Dequeue\n3. Peek\n4. Display\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(&q, value);

break;

case 2:

value = dequeue(&q);

if (value != -1) {

printf("Dequeued: %d\n", value);

}

break;

case 3:

value = peek(&q);

if (value != -1) {

printf("Front element: %d\n", value);

}

break;

case 4:

printf("Queue contents: ");

display(&q);

break;

case 5:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Please try again.\n");

} } }

...............................................................................................................

PRACTICAL 8

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

int queue[MAX];

int front = -1;

int rear = -1;

// Check if the queue is empty

int isEmpty() {

return front == -1;

}

// Check if the queue is full

int isFull() {

return (rear + 1) % MAX == front;

}

// Add an element to the queue

void enqueue(int value) {

if (isFull()) {

printf("Queue is full! Cannot enqueue.\n");

return;

}

if (isEmpty()) {

front = rear = 0;

} else {

rear = (rear + 1) % MAX;

}

queue[rear] = value;

printf("Enqueued %d into the queue.\n", value);

}

// Remove an element from the queue

int dequeue() {

if (isEmpty()) {

printf("Queue is empty! Cannot dequeue.\n");

return -1;

}

int value = queue[front];

if (front == rear) { // Queue becomes empty

front = rear = -1;

} else {

front = (front + 1) % MAX;

}

return value;

}

// Display the front and rear elements of the queue

void displayFrontRear() {

if (isEmpty()) {

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

} else {

printf("Front element: %d\n", queue[front]);

printf("Rear element: %d\n", queue[rear]);

}

}

// Display all elements in the queue

void display() {

if (isEmpty()) {

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

return;

}

int i = front;

printf("Queue elements: ");

while (i != rear) {

printf("%d ", queue[i]);

i = (i + 1) % MAX;

}

printf("%d\n", queue[rear]);

}

int main() {

int choice, value;

while (1) {

printf("\n1. Enqueue\n2. Dequeue\n3. Display Front and Rear\n4. Display Queue\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

value = dequeue();

if (value != -1) {

printf("Dequeued value: %d\n", value);

}

break;

case 3:

displayFrontRear();

break;

case 4:

display();

break;

case 5:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Try again.\n");

}

}

}

................................................................................................................................

PRACTICAL 9

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* front = NULL;

struct Node\* rear = NULL;

void enqueue(int value) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

return;

}

newNode->data = value;

if (front == NULL) {

front = rear = newNode;

rear->next = front; // Circular linking

} else {

rear->next = newNode;

rear = newNode;

rear->next = front; // Maintain circular structure

}

printf("Enqueued %d into the queue.\n", value);

}

int dequeue() {

if (front == NULL) {

printf("Queue is empty! Cannot dequeue.\n");

return -1;

}

int value = front->data;

if (front == rear) { // Single element in the queue

free(front);

front = rear = NULL;

} else {

struct Node\* temp = front;

front = front->next;

rear->next = front; // Maintain circular structure

free(temp);

}

return value;

}

void displayFrontRear() {

if (front == NULL) {

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

} else {

printf("Front element: %d\n", front->data);

printf("Rear element: %d\n", rear->data);

}

}

void display() {

if (front == NULL) {

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

return;

}

struct Node\* temp = front;

printf("Queue elements: ");

do {

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

temp = temp->next;

} while (temp != front);

printf("\n");

}

int main() {

int choice, value;

while (1) {

printf("\n1. Enqueue\n2. Dequeue\n3. Display Front and Rear\n4. Display Queue\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

value = dequeue();

if (value != -1) {

printf("Dequeued value: %d\n", value);

}

break;

case 3:

displayFrontRear();

break;

case 4:

display();

break;

case 5:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Try again.\n");

}

}

}

..........................................................................................................................

PRACTICAL 10

#include <stdio.h>

#include <stdlib.h>

#define MAX 5 // Maximum size of the deque

int deque[MAX];

int front = -1;

int rear = -1;

// Check if the deque is empty

int isEmpty() {

return front == -1;

}

// Check if the deque is full

int isFull() {

return (rear + 1) % MAX == front;

}

// Insert an element at the front

void insertFront(int value) {

if (isFull()) {

printf("Deque is full! Cannot insert at front.\n");

return;

}

if (isEmpty()) {

front = rear = 0;

} else {

front = (front - 1 + MAX) % MAX; // Circular decrement

}

deque[front] = value;

printf("Inserted %d at the front.\n", value);

}

// Insert an element at the rear

void insertRear(int value) {

if (isFull()) {

printf("Deque is full! Cannot insert at rear.\n");

return;

}

if (isEmpty()) {

front = rear = 0;

} else {

rear = (rear + 1) % MAX; // Circular increment

}

deque[rear] = value;

printf("Inserted %d at the rear.\n", value);

}

// Delete an element from the front

int deleteFront() {

if (isEmpty()) {

printf("Deque is empty! Cannot delete from front.\n");

return -1;

}

int value = deque[front];

if (front == rear) { // Only one element left

front = rear = -1;

} else {

front = (front + 1) % MAX; // Circular increment

}

return value;

}

// Delete an element from the rear

int deleteRear() {

if (isEmpty()) {

printf("Deque is empty! Cannot delete from rear.\n");

return -1;

}

int value = deque[rear];

if (front == rear) { // Only one element left

front = rear = -1;

} else {

rear = (rear - 1 + MAX) % MAX; // Circular decrement

}

return value;

}

// Display the front and rear elements

void displayFrontRear() {

if (isEmpty()) {

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

} else {

printf("Front element: %d\n", deque[front]);

printf("Rear element: %d\n", deque[rear]);

}

}

// Display all elements in the deque

void display() {

if (isEmpty()) {

printf("Deque is empty.\n");

return;

}

int i = front;

printf("Deque elements: ");

while (i != rear) {

printf("%d ", deque[i]);

i = (i + 1) % MAX; // Circular increment

}

printf("%d\n", deque[rear]); // Print the rear element

}

int main() {

int choice, value;

while (1) {

printf("\n1. Insert at Front\n2. Insert at Rear\n3. Delete from Front\n4. Delete from Rear\n5. Display Front and Rear\n6. Display Deque\n7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert at front: ");

scanf("%d", &value);

insertFront(value);

break;

case 2:

printf("Enter value to insert at rear: ");

scanf("%d", &value);

insertRear(value);

break;

case 3:

value = deleteFront();

if (value != -1) {

printf("Deleted value from front: %d\n", value);

}

break;

case 4:

value = deleteRear();

if (value != -1) {

printf("Deleted value from rear: %d\n", value);

}

break;

case 5:

displayFrontRear();

break;

case 6:

display();

break;

case 7:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Try again.\n");

}

}

}

.......................................................................................................................

PRACTICAL 11

#include <stdio.h>

#include <stdlib.h>

#define MAX 5 // Maximum size of the queue

// Stacks used to implement the queue

int stack1[MAX];

int stack2[MAX];

int top1 = -1; // Top pointer for stack1

int top2 = -1; // Top pointer for stack2

// Function to check if a stack is empty

int isEmpty(int top) {

return top == -1;

}

// Function to check if a stack is full

int isFull(int top) {

return top == MAX - 1;

}

// Function to push a value onto a stack

void push(int stack[], int \*top, int value) {

if (isFull(\*top)) {

printf("Stack is full! Cannot push %d.\n", value);

return;

}

stack[++(\*top)] = value;

}

// Function to pop a value from a stack

int pop(int stack[], int \*top) {

if (isEmpty(\*top)) {

printf("Stack is empty! Cannot pop.\n");

return -1;

}

return stack[(\*top)--];

}

// Function to enqueue an element into the queue

void enqueue(int value) {

if (isFull(top1)) {

printf("Queue is full! Cannot enqueue %d.\n", value);

return;

}

push(stack1, &top1, value);

printf("Enqueued %d into the queue.\n", value);

}

// Function to dequeue an element from the queue

int dequeue() {

if (isEmpty(top2)) {

if (isEmpty(top1)) {

printf("Queue is empty! Cannot dequeue.\n");

return -1;

}

// Transfer elements from stack1 to stack2 to maintain queue order

while (!isEmpty(top1)) {

push(stack2, &top2, pop(stack1, &top1));

}

}

return pop(stack2, &top2);

}

// Function to display the front and rear elements of the queue

void displayFrontRear() {

if (isEmpty(top1) && isEmpty(top2)) {

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

} else {

// Front element is the top of stack2 or the bottom of stack1

if (!isEmpty(top2)) {

printf("Front element: %d\n", stack2[top2]);

} else {

printf("Front element: %d\n", stack1[0]);

}

// Rear element is the top of stack1 or the bottom of stack2

if (!isEmpty(top1)) {

printf("Rear element: %d\n", stack1[top1]);

} else {

printf("Rear element: %d\n", stack2[0]);

}

}

}

// Function to display all elements in the queue

void display() {

if (isEmpty(top1) && isEmpty(top2)) {

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

return;

}

printf("Queue elements: ");

// Display elements from stack2 (in reverse order)

for (int i = top2; i >= 0; i--) {

printf("%d ", stack2[i]);

}

// Display elements from stack1 (in original order)

for (int i = 0; i <= top1; i++) {

printf("%d ", stack1[i]);

}

printf("\n");

}

// Main function with a menu-driven interface

int main() {

int choice, value;

while (1) {

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

printf("2. Dequeue\n");

printf("3. Display Front and Rear\n");

printf("4. Display Queue\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

value = dequeue();

if (value != -1) {

printf("Dequeued value: %d\n", value);

}

break;

case 3:

displayFrontRear();

break;

case 4:

display();

break;

case 5:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Try again.\n");

}

}

}

.............................................................................................................................

PRACTICAL 12

#include <stdio.h>

#include <stdlib.h>

// Node structure for the Binary Search Tree

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int data) {

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

if (!newNode) {

printf("Memory allocation failed\n");

return NULL;

}

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

// Function to insert a value into the BST

struct Node\* insert(struct Node\* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

} else if (data > root->data) {

root->right = insert(root->right, data);

} else {

printf("Duplicate value not allowed\n");

}

return root;

}

// Function to find the node with the smallest value in a subtree

struct Node\* findMin(struct Node\* root) {

while (root && root->left != NULL) {

root = root->left;

}

return root;

}

// Function to delete a node from the BST

struct Node\* deleteNode(struct Node\* root, int data) {

if (root == NULL) {

printf("Element not found\n");

return NULL;

}

if (data < root->data) {

root->left = deleteNode(root->left, data);

} else if (data > root->data) {

root->right = deleteNode(root->right, data);

} else {

// Node with one or no children

if (root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

// Node with two children

struct Node\* temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

// Function for Inorder Traversal

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

// Function for Preorder Traversal

void preorder(struct Node\* root) {

if (root != NULL) {

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

preorder(root->left);

preorder(root->right);

}

}

// Function for Postorder Traversal

void postorder(struct Node\* root) {

if (root != NULL) {

postorder(root->left);

postorder(root->right);

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

}

}

int main() {

struct Node\* root = NULL;

int choice, value;

while (1) {

printf("\n1. Insert\n2. Delete\n3. Inorder Traversal\n4. Preorder Traversal\n5. Postorder Traversal\n6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

root = insert(root, value);

printf("Inserted %d into the BST.\n", value);

break;

case 2:

printf("Enter value to delete: ");

scanf("%d", &value);

root = deleteNode(root, value);

printf("Deleted %d from the BST (if it existed).\n", value);

break;

case 3:

printf("Inorder Traversal: ");

inorder(root);

printf("\n");

break;

case 4:

printf("Preorder Traversal: ");

preorder(root);

printf("\n");

break;

case 5:

printf("Postorder Traversal: ");

postorder(root);

printf("\n");

break;

case 6:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice! Please try again.\n");

}

}

}