|  |
| --- |
| **SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES** |
| **COMPUTER SCIENCE AND ENGINEERING PROGRAMME** |

**SUB CODE: CSA0392 SUB NAME: Data Structures for Hashing Techniques**

**LIST OF PROGRAMS**

**DAY 5 : 29.07.2024**

**Lab Questions to be practiced with test cases**

1. Write a program that implement Queue (its operations) using Arrays.

Answer:

#include <stdio.h>

#include <stdlib.h>

#define MAX 100 // Define the maximum size of the queue

// Define the queue structure

typedef struct {

int arr[MAX];

int front;

int rear;

int size;

} Queue;

// Function to initialize the queue

void initializeQueue(Queue\* queue) {

queue->front = 0;

queue->rear = -1;

queue->size = 0;

}

// Function to check if the queue is empty

int isEmpty(Queue\* queue) {

return queue->size == 0;

}

// Function to check if the queue is full

int isFull(Queue\* queue) {

return queue->size == MAX;

}

// Function to add an element to the end of the queue

void enqueue(Queue\* queue, int value) {

if (isFull(queue)) {

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

return;

}

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

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

queue->size++;

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

}

// Function to remove an element from the front of the queue

int dequeue(Queue\* queue) {

if (isEmpty(queue)) {

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

return -1; // Return -1 to indicate an error

}

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

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

queue->size--;

return value;

}

// Function to get the element at the front of the queue without removing it

int peek(Queue\* queue) {

if (isEmpty(queue)) {

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

return -1; // Return -1 to indicate an error

}

return queue->arr[queue->front];

}

// Function to display the elements of the queue

void displayQueue(Queue\* queue) {

if (isEmpty(queue)) {

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

return;

}

printf("Queue elements are:\n");

int i;

for (i = 0; i < queue->size; i++) {

printf("%d ", queue->arr[(queue->front + i) % MAX]);

}

printf("\n");

}

int main() {

Queue queue;

initializeQueue(&queue);

int choice, value;

while (1) {

printf("\nQueue Menu:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Peek\n");

printf("4. Display\n");

printf("5. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to enqueue: ");

scanf("%d", &value);

enqueue(&queue, value);

break;

case 2:

value = dequeue(&queue);

if (value != -1) {

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

}

break;

case 3:

value = peek(&queue);

if (value != -1) {

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

}

break;

case 4:

displayQueue(&queue);

break;

case 5:

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

exit(0);

default:

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

}

}

1. Write a program that implement Queue (its operations) using Linked list(Pointers).

Answer:

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the linked list

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Define the structure for the queue

typedef struct Queue {

Node\* front;

Node\* rear;

} Queue;

// Function to initialize the queue

void initializeQueue(Queue\* queue) {

queue->front = NULL;

queue->rear = NULL;

}

// Function to check if the queue is empty

int isEmpty(Queue\* queue) {

return queue->front == NULL;

}

// Function to add an element to the end of the queue

void enqueue(Queue\* queue, int value) {

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

if (newNode == NULL) {

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

return;

}

newNode->data = value;

newNode->next = NULL;

if (isEmpty(queue)) {

queue->front = newNode;

} else {

queue->rear->next = newNode;

}

queue->rear = newNode;

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

}

// Function to remove an element from the front of the queue

int dequeue(Queue\* queue) {

if (isEmpty(queue)) {

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

return -1; // Return -1 to indicate an error

}

Node\* temp = queue->front;

int value = temp->data;

queue->front = queue->front->next;

if (queue->front == NULL) {

queue->rear = NULL;

}

free(temp);

return value;

}

// Function to get the element at the front of the queue without removing it

int peek(Queue\* queue) {

if (isEmpty(queue)) {

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

return -1; // Return -1 to indicate an error

}

return queue->front->data;

}

// Function to display the elements of the queue

void displayQueue(Queue\* queue) {

if (isEmpty(queue)) {

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

return;

}

printf("Queue elements are:\n");

Node\* temp = queue->front;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

Queue queue;

initializeQueue(&queue);

int choice, value;

while (1) {

printf("\nQueue Menu:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Peek\n");

printf("4. Display\n");

printf("5. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to enqueue: ");

scanf("%d", &value);

enqueue(&queue, value);

break;

case 2:

value = dequeue(&queue);

if (value != -1) {

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

}

break;

case 3:

value = peek(&queue);

if (value != -1) {

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

}

break;

case 4:

displayQueue(&queue);

break;

case 5:

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

exit(0);

default:

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

}

}

return 0;

}