

## Day 5

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

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

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

```
#define MAX 5
```

```
struct Queue {
```

```
    int items[MAX];
```

```
    int front;
```

```
    int rear;
```

```
};
```

```
struct Queue* createQueue() {
```

```
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
```

```
    q->front = -1;
```

```
    q->rear = -1;
```

```
    return q;
```

```
}
```

```
int isFull(struct Queue* q) {
```

```
    return q->rear == MAX - 1;
```

```
}
```

```
int isEmpty(struct Queue* q) {  
    return q->front == -1 || q->front > q->rear;  
}
```

```
void enqueue(struct Queue* q, int value) {  
    if (isFull(q)) {  
        printf("Queue is full!\n");  
        return;  
    }  
    if (q->front == -1) {  
        q->front = 0;  
    }  
    q->rear++;  
    q->items[q->rear] = value;  
    printf("%d enqueued to queue\n", value);  
}
```

```
int dequeue(struct Queue* q) {  
    if (isEmpty(q)) {  
        printf("Queue is empty!\n");  
        return -1;  
    }  
    int item = q->items[q->front];  
    q->front++;  
    if (q->front > q->rear) {
```

```
        q->front = q->rear = -1;
    }
    return item;
}
```

```
void display(struct 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->items[i]);
    }
    printf("\n");
}
```

```
int main() {
    struct Queue* q = createQueue();
    enqueue(q, 10);
    enqueue(q, 20);
    enqueue(q, 30);
    display(q);
    printf("%d dequeued from queue\n", dequeue(q));
    display(q);
}
```

```

        return 0;
    }
}

```

Output:

The screenshot shows a C code editor with a file named 'main.c'. The code implements a queue using an array. It includes functions for enqueue, dequeue, and display. The main function tests these operations. The output window shows the results of the execution.

```

main.c
50     return item;
51 }
52
53 void display(struct Queue* q) {
54     if (isEmpty(q)) {
55         printf("Queue is empty!\n");
56         return;
57     }
58     printf("Queue elements: ");
59     for (int i = q->front; i <= q->rear; i++) {
60         printf("%d ", q->items[i]);
61     }
62     printf("\n");
63 }
64
65 int main() {
66     struct Queue* q = createQueue();
67     enqueue(q, 10);
68     enqueue(q, 20);
69     enqueue(q, 30);
70     display(q);
71     printf("%d dequeued from queue\n", dequeue(q));
72     display(q);
73     return 0;
74 }
75
Output
/tmp/OQvK06o0rp.o
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
Queue elements: 10 20 30
10 dequeued from queue
Queue elements: 20 30

=== Code Execution Successful ===

```

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

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

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

```
// Node structure
```

```
typedef struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
} Node;
```

```
// Queue structure
```

```
typedef struct Queue {
```

```
    Node* front;
```

```
    Node* rear;
```

```
} Queue;
```

```
// Function to create a new node
```

```
Node* createNode(int data) {
```

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

```
    newNode->data = data;
```

```
    newNode->next = NULL;
```

```
    return newNode;
```

```
}
```

```
// Function to create a new queue
```

```
Queue* createQueue() {
```

```
    Queue* q = (Queue*)malloc(sizeof(Queue));
```

```
    q->front = NULL;
```

```
    q->rear = NULL;
```

```
    return q;
```

```
}
```

```
// Function to enqueue an element
```

```
void enqueue(Queue* q, int data) {
```

```
    Node* newNode = createNode(data);
```

```
    if (q->rear == NULL) {
```

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

    } else {

        q->rear->next = newNode;

        q->rear = newNode;

    }

}
```

// Function to dequeue an element

```
int dequeue(Queue* q) {

    if (q->front == NULL) {

        printf("Queue is empty\n");

        return -1;

    } else {

        Node* temp = q->front;

        int data = temp->data;

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

        if (q->front == NULL) {

            q->rear = NULL;

        }

        free(temp);

        return data;

    }

}
```


// Function to print the queue

```
void printQueue(Queue* q) {  
    Node* temp = q->front;  
    while (temp != NULL) {  
        printf("%d ", temp->data);  
        temp = temp->next;  
    }  
    printf("\n");  
}
```

```
int main() {  
    Queue* q = createQueue();  
  
    enqueue(q, 1);  
    enqueue(q, 2);  
    enqueue(q, 3);  
    enqueue(q, 4);  
    enqueue(q, 5);  
  
    printQueue(q); // Output: 1 2 3 4 5  
  
    int dequeued = dequeue(q);  
    printf("Dequeued: %d\n", dequeued); // Output: 1  
  
    printQueue(q); // Output: 2 3 4 5
```

```
    return 0;
}
```

output:

main.c	Run	Output
<pre>63- while (temp != NULL) { 64-     printf("%d ", temp-&gt;data); 65-     temp = temp-&gt;next; 66- } 67- printf("\n"); 68- } 69- 70- int main() { 71-     Queue* q = createQueue(); 72- 73-     enqueue(q, 1); 74-     enqueue(q, 2); 75-     enqueue(q, 3); 76-     enqueue(q, 4); 77-     enqueue(q, 5); 78- 79-     printQueue(q); // Output: 1 2 3 4 5 80- 81-     int dequeued = dequeue(q); 82-     printf("Dequeued: %d\n", dequeued); // Output: 1 83- 84-     printQueue(q); // Output: 2 3 4 5 85- }</pre>		<pre>/tmp/gKns3fy8hA.o 1 2 3 4 5 Dequeued: 1 2 3 4 5  === Code Execution Successful ===</pre>