Queue:

What is queue?

→ A queue is yet another data structure. It works on the principle of First In First Out (FIFO).

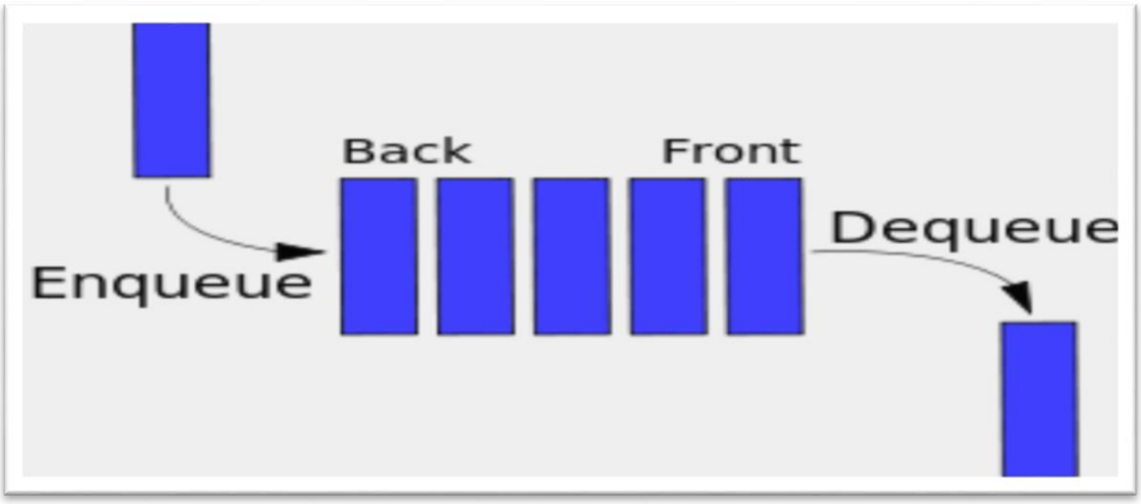
Ex: - You are standing in a queue outside your school canteen; the first person standing in the line would be the first person to get the food.

Basic Operations:

- To add an element at the end is called Enqueue.
- Removing an element from the front is called Dequeue.
- To check if the Queue is empty or not IsEmpty.
- To check if the Queue is full or not IsFull.

In queue we basically use two pointers Front and Rear both initially set to -1 when the queue is empty. Front and Rear keeps the track of the first and the last element respectively.

When the first element is added the value of front is set to 0 and the value of Rear keeps increasing by 1 as new elements get added. We know in Dequeue elements are removed from front so during dequeue the value of Front is incremented by 1.

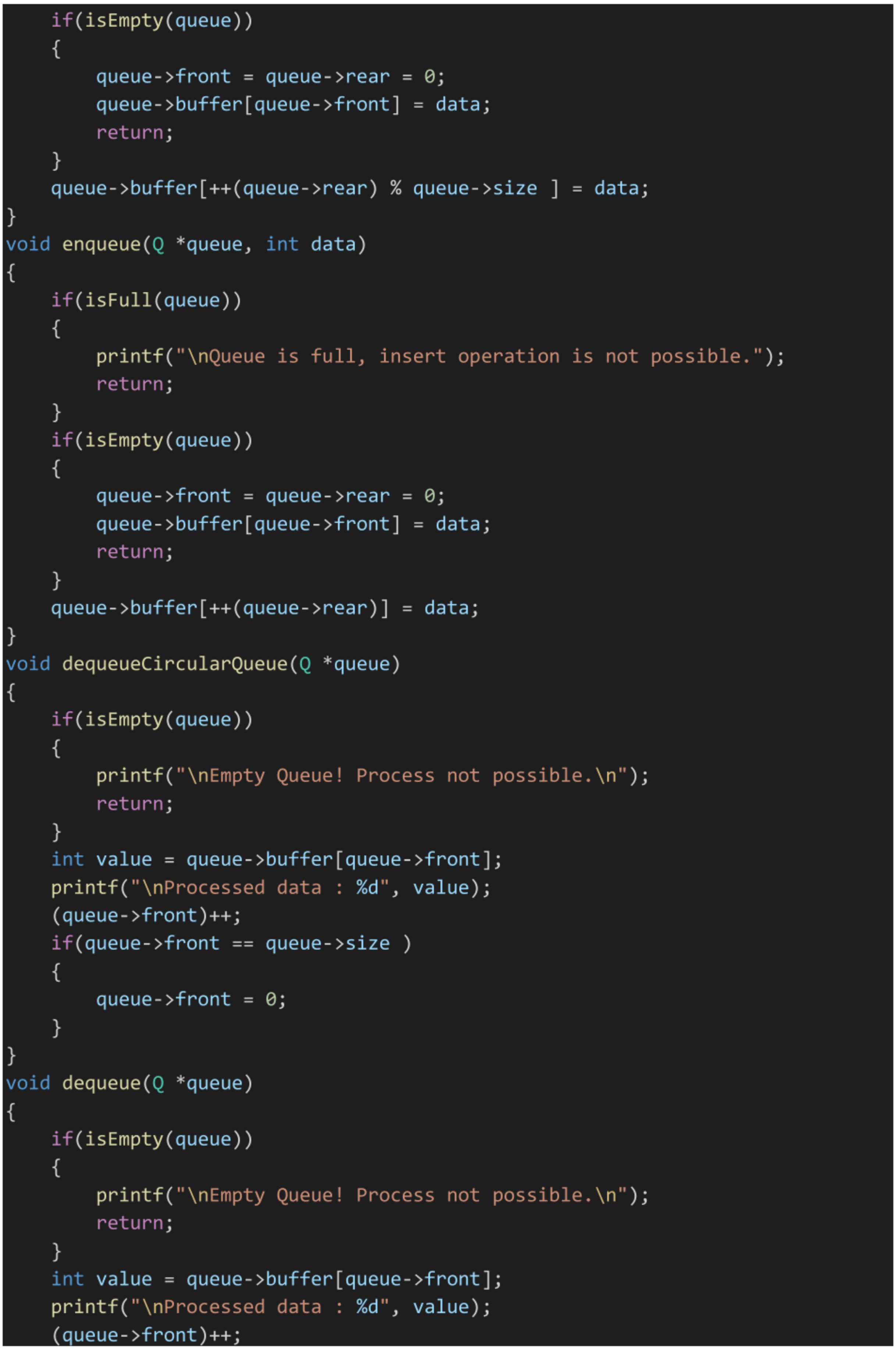


Types of Queue:

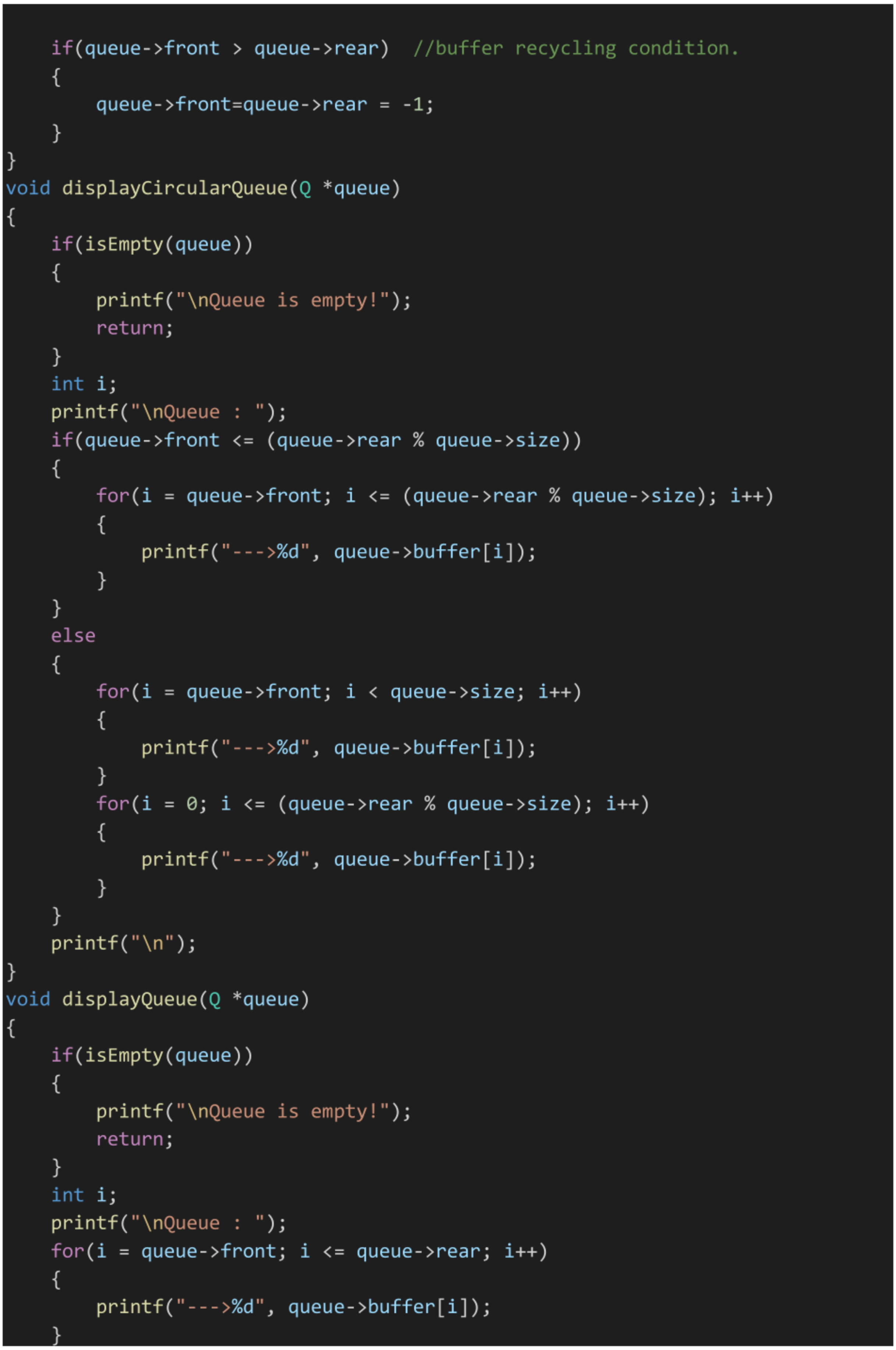
- Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue

Implementation of Linear and Circular Queue

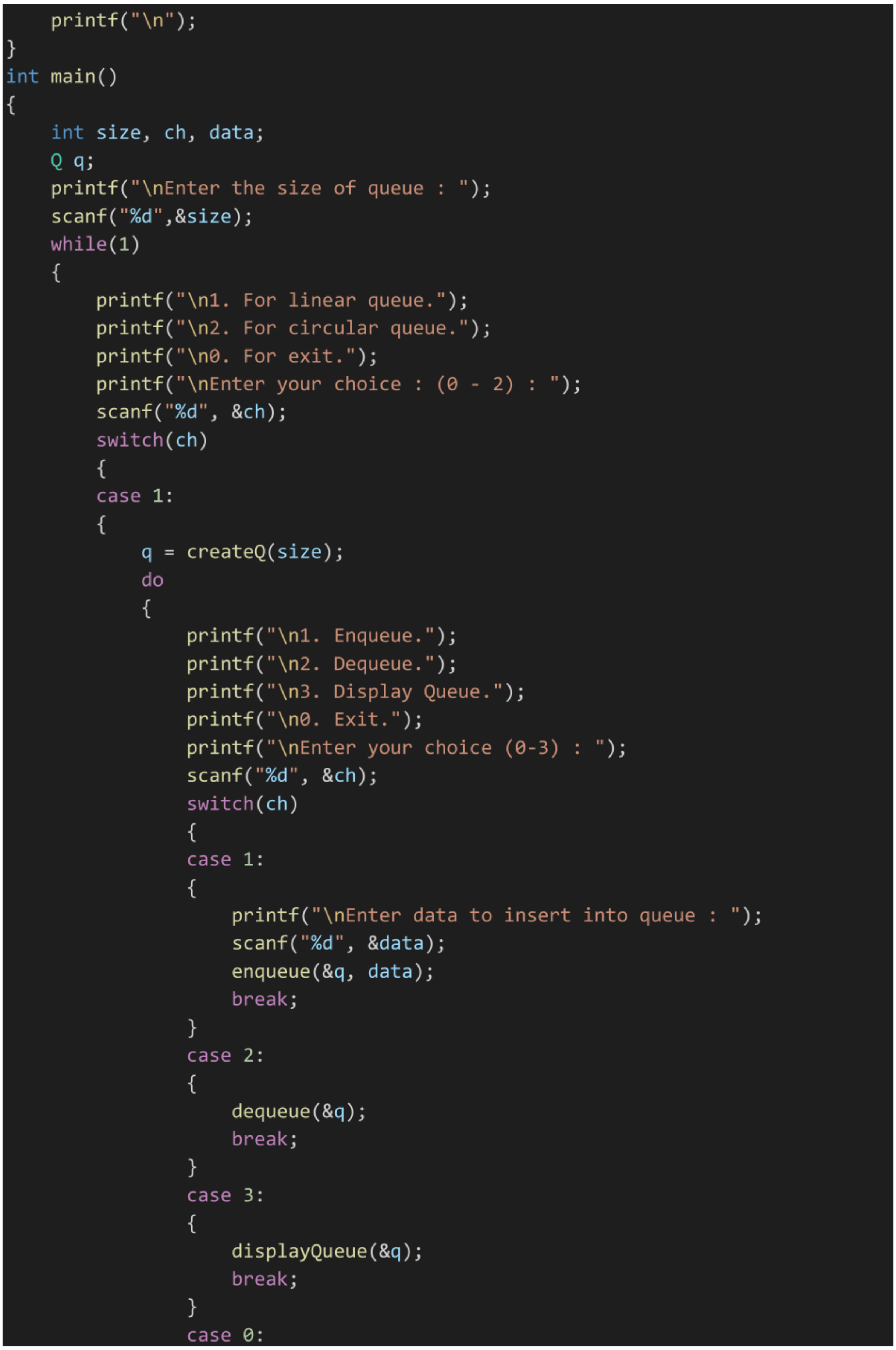
```c
#include<stdio.h>
struct Queue
{
    int *buffer;
    int front, rear, size;
};
typedef struct Queue Q;
Q createQ(int size)
{
    Q queue;
    queue.buffer = (int*)malloc(sizeof(int)*size);
    queue.front = -1;
    queue.rear = -1;
    queue.size = size;
    return queue;
}

int isEmpty(Q *queue)
{
    return queue->front == -1;
}
int isFull(Q *queue)
{
    return queue->rear == queue->size - 1;
}
int isCircularQueueFull(Q *q)
{
    return ((q->front == 0 && q->rear==q->size - 1) || ((q->rear%q->size) + 1
== q->front));
}
void enqueueCircularQueue(Q *queue, int data)
{
    if(isCircularQueueFull(queue))
    {
        printf("\nQueue is full, insert operation is not possible.");
        return;
    }
```



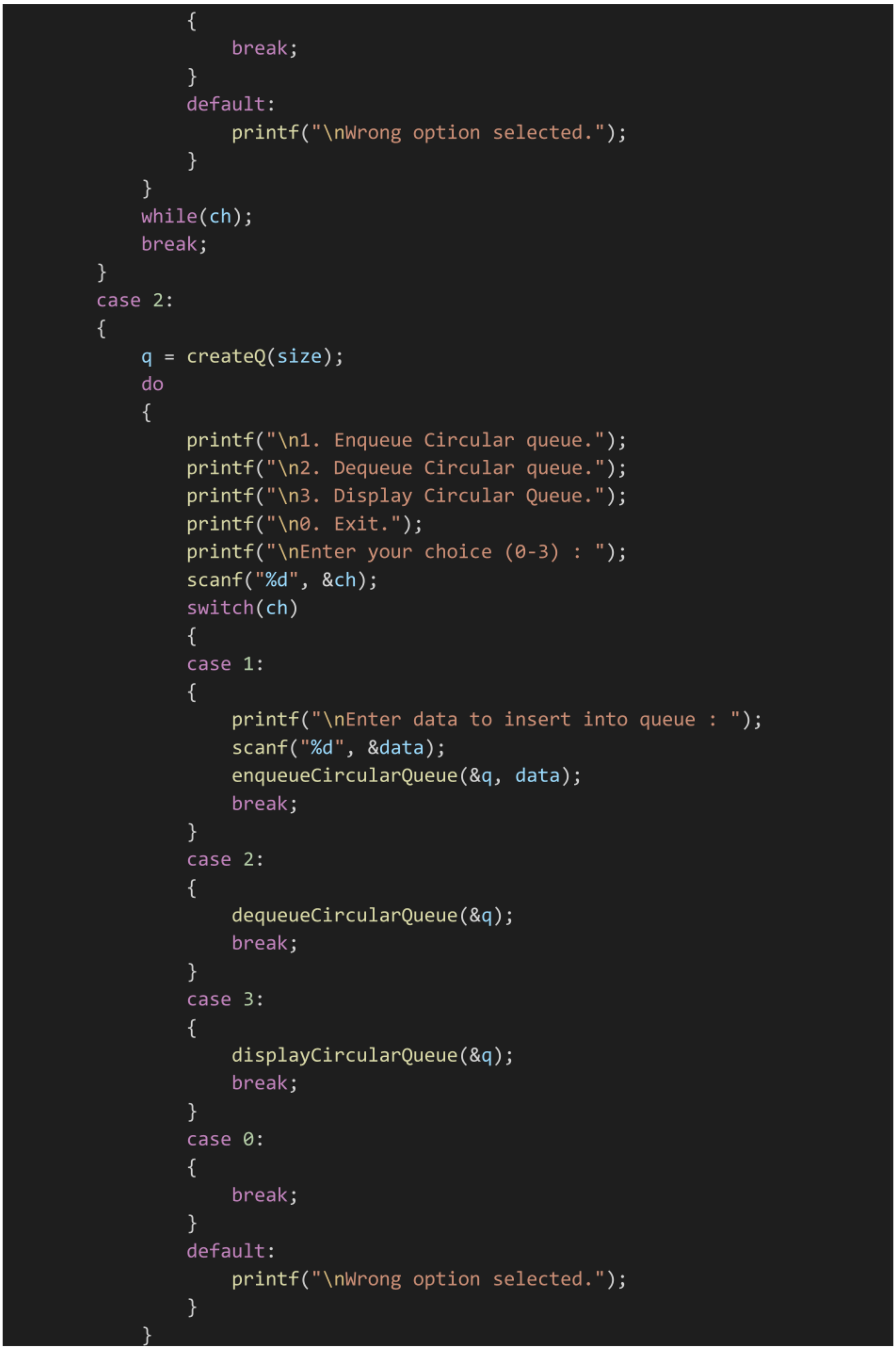
```c
    if(isEmpty(queue))
    {
        queue->front = queue->rear = 0;
        queue->buffer[queue->front] = data;
        return;
    }
    queue->buffer[++(queue->rear) % queue->size ] = data;
}
void enqueue(Q *queue, int data)
{
    if(isFull(queue))
    {
        printf("\nQueue is full, insert operation is not possible.");
        return;
    }
    if(isEmpty(queue))
    {
        queue->front = queue->rear = 0;
        queue->buffer[queue->front] = data;
        return;
    }
    queue->buffer[++(queue->rear)] = data;
}
void dequeueCircularQueue(Q *queue)
{
    if(isEmpty(queue))
    {
        printf("\nEmpty Queue! Process not possible.\n");
        return;
    }
    int value = queue->buffer[queue->front];
    printf("\nProcessed data : %d", value);
    (queue->front)++;
    if(queue->front == queue->size )
    {
        queue->front = 0;
    }
}
void dequeue(Q *queue)
{
    if(isEmpty(queue))
    {
        printf("\nEmpty Queue! Process not possible.\n");
        return;
    }
    int value = queue->buffer[queue->front];
    printf("\nProcessed data : %d", value);
    (queue->front)++;
```



```c
    if(queue->front > queue->rear)  //buffer recycling condition.
    {
        queue->front=queue->rear = -1;
    }
}
void displayCircularQueue(Q *queue)
{
    if(isEmpty(queue))
    {
        printf("\nQueue is empty!");
        return;
    }
    int i;
    printf("\nQueue : ");
    if(queue->front <= (queue->rear % queue->size))
    {
        for(i = queue->front; i <= (queue->rear % queue->size); i++)
        {
            printf("--->%d", queue->buffer[i]);
        }
    }
    else
    {
        for(i = queue->front; i < queue->size; i++)
        {
            printf("--->%d", queue->buffer[i]);
        }
        for(i = 0; i <= (queue->rear % queue->size); i++)
        {
            printf("--->%d", queue->buffer[i]);
        }
    }
    printf("\n");
}
void displayQueue(Q *queue)
{
    if(isEmpty(queue))
    {
        printf("\nQueue is empty!");
        return;
    }
    int i;
    printf("\nQueue : ");
    for(i = queue->front; i <= queue->rear; i++)
    {
        printf("--->%d", queue->buffer[i]);
    }
```



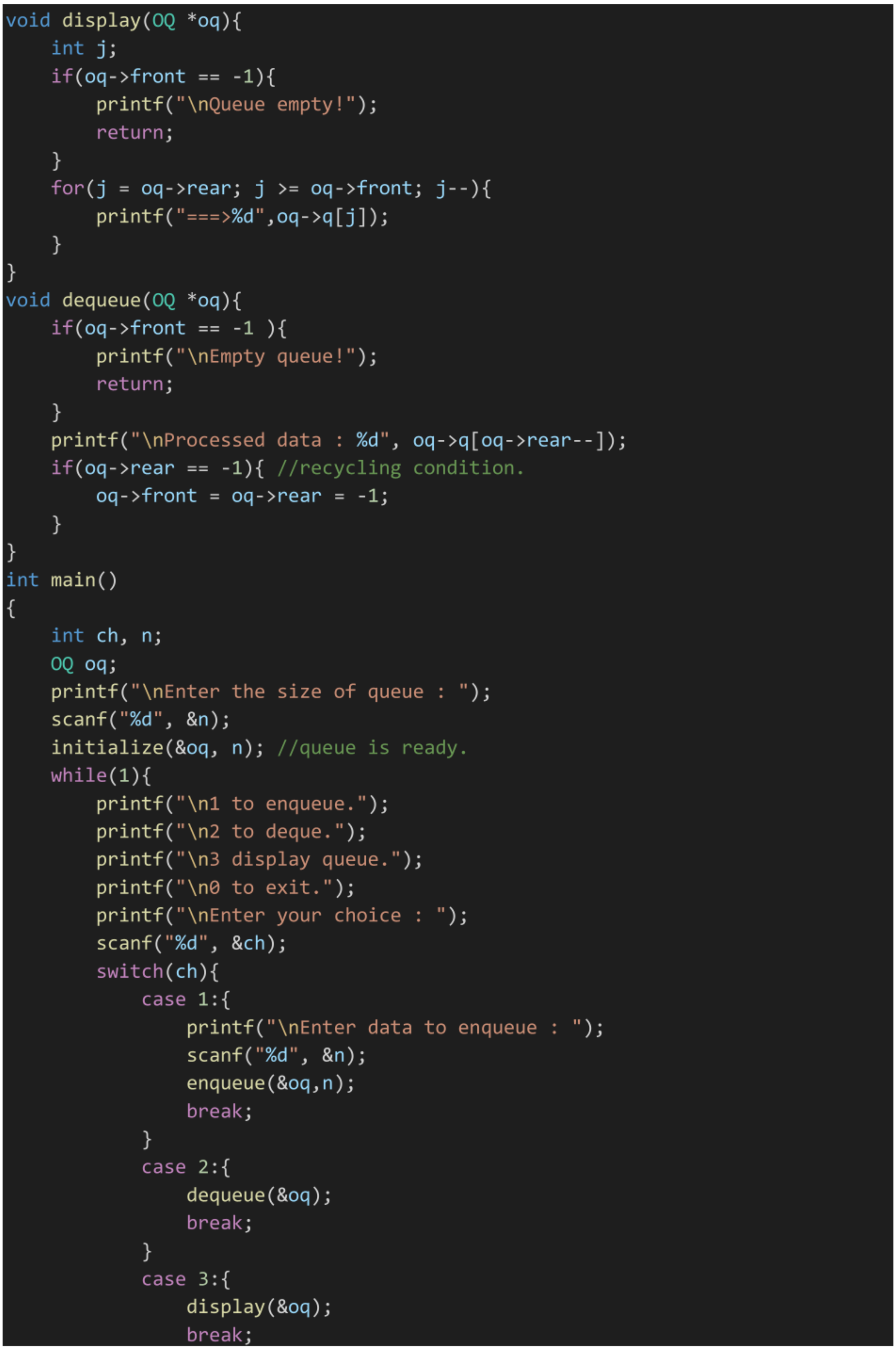
```c
    printf("\n");
}
int main()
{
    int size, ch, data;
    Q q;
    printf("\nEnter the size of queue : ");
    scanf("%d",&size);
    while(1)
    {
        printf("\n1. For linear queue.");
        printf("\n2. For circular queue.");
        printf("\n0. For exit.");
        printf("\nEnter your choice : (0 - 2) : ");
        scanf("%d", &ch);
        switch(ch)
        {
        case 1:
        {
            q = createQ(size);
            do
            {
                printf("\n1. Enqueue.");
                printf("\n2. Dequeue.");
                printf("\n3. Display Queue.");
                printf("\n0. Exit.");
                printf("\nEnter your choice (0-3) : ");
                scanf("%d", &ch);
                switch(ch)
                {
                case 1:
                {
                    printf("\nEnter data to insert into queue : ");
                    scanf("%d", &data);
                    enqueue(&q, data);
                    break;
                }
                case 2:
                {
                    dequeue(&q);
                    break;
                }
                case 3:
                {
                    displayQueue(&q);
                    break;
                }
                case 0:
```



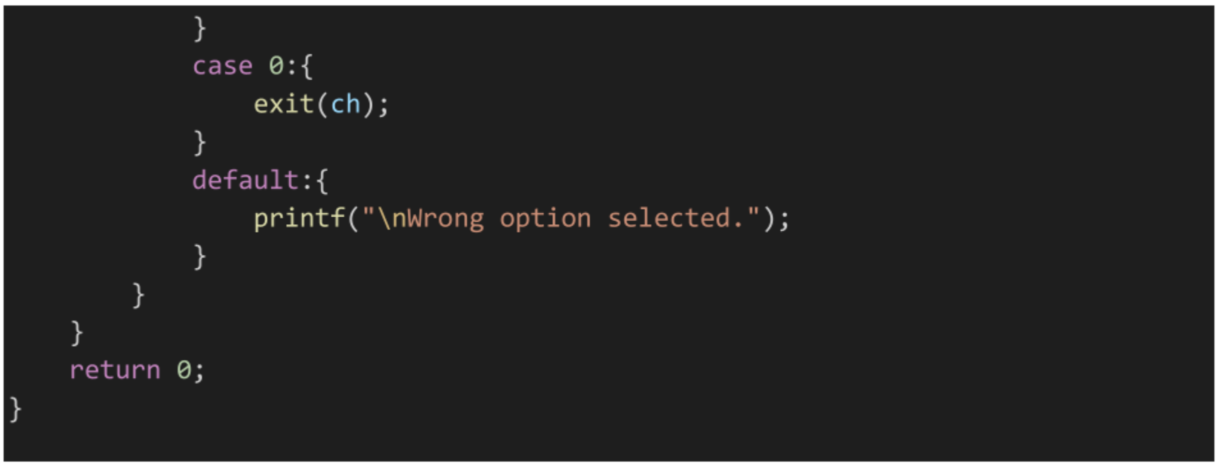
```c
                {
                    break;
                }
                default:
                    printf("\nWrong option selected.");
                }
            }
            while(ch);
            break;
        }
        case 2:
        {
            q = createQ(size);
            do
            {
                printf("\n1. Enqueue Circular queue.");
                printf("\n2. Dequeue Circular queue.");
                printf("\n3. Display Circular Queue.");
                printf("\n0. Exit.");
                printf("\nEnter your choice (0-3) : ");
                scanf("%d", &ch);
                switch(ch)
                {
                case 1:
                {
                    printf("\nEnter data to insert into queue : ");
                    scanf("%d", &data);
                    enqueueCircularQueue(&q, data);
                    break;
                }
                case 2:
                {
                    dequeueCircularQueue(&q);
                    break;
                }
                case 3:
                {
                    displayCircularQueue(&q);
                    break;
                }
                case 0:
                {
                    break;
                }
                default:
                    printf("\nWrong option selected.");
                }
            }
            while(ch);
            break;
        }
        case 0:
            exit(ch);
        }
    }
    return 0;
}
```

Implementation of Priority Queue

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

struct OrderedQueue{
    int *q;
    int front;
    int rear;
    int size;
};
typedef struct OrderedQueue OQ;
void initialize(OQ *oq, int ne){
    oq->q = (int*)malloc(sizeof(int) * ne);
    oq->size = ne;
    oq->front = oq->rear = -1;
}
void enqueue(OQ *oq, int data){
    int j;
    if(oq->rear == oq->size - 1){
        printf("\nQueue is full.");
        return;
    }
    if(oq->front == -1){
        oq->front = oq->rear = 0;
        oq->q[oq->front] = data;
    }
    else{
        for(j = oq->rear; j > -1 && oq->q[j] > data; j--){
            oq->q[j + 1] = oq->q[j];
        }
        oq->q[j+1] = data;
        oq->rear++;
    }
    printf("\nData inserted successfully.");
}
```



```c
void display(OQ *oq){
    int j;
    if(oq->front == -1){
        printf("\nQueue empty!");
        return;
    }
    for(j = oq->rear; j >= oq->front; j--){
        printf("===>%d",oq->q[j]);
    }
}
void dequeue(OQ *oq){
    if(oq->front == -1 ){
        printf("\nEmpty queue!");
        return;
    }
    printf("\nProcessed data : %d", oq->q[oq->rear--]);
    if(oq->rear == -1){ //recycling condition.
        oq->front = oq->rear = -1;
    }
}
int main()
{
    int ch, n;
    OQ oq;
    printf("\nEnter the size of queue : ");
    scanf("%d", &n);
    initialize(&oq, n); //queue is ready.
    while(1){
        printf("\n1 to enqueue.");
        printf("\n2 to deque.");
        printf("\n3 display queue.");
        printf("\n0 to exit.");
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch(ch){
            case 1:{
                printf("\nEnter data to enqueue : ");
                scanf("%d", &n);
                enqueue(&oq,n);
                break;
            }
            case 2:{
                dequeue(&oq);
                break;
            }
            case 3:{
                display(&oq);
                break;
```


```
            }
            case 0:{
                exit(ch);
            }
            default:{
                printf("\nWrong option selected.");
            }
        }
    }
    return 0;
}
```