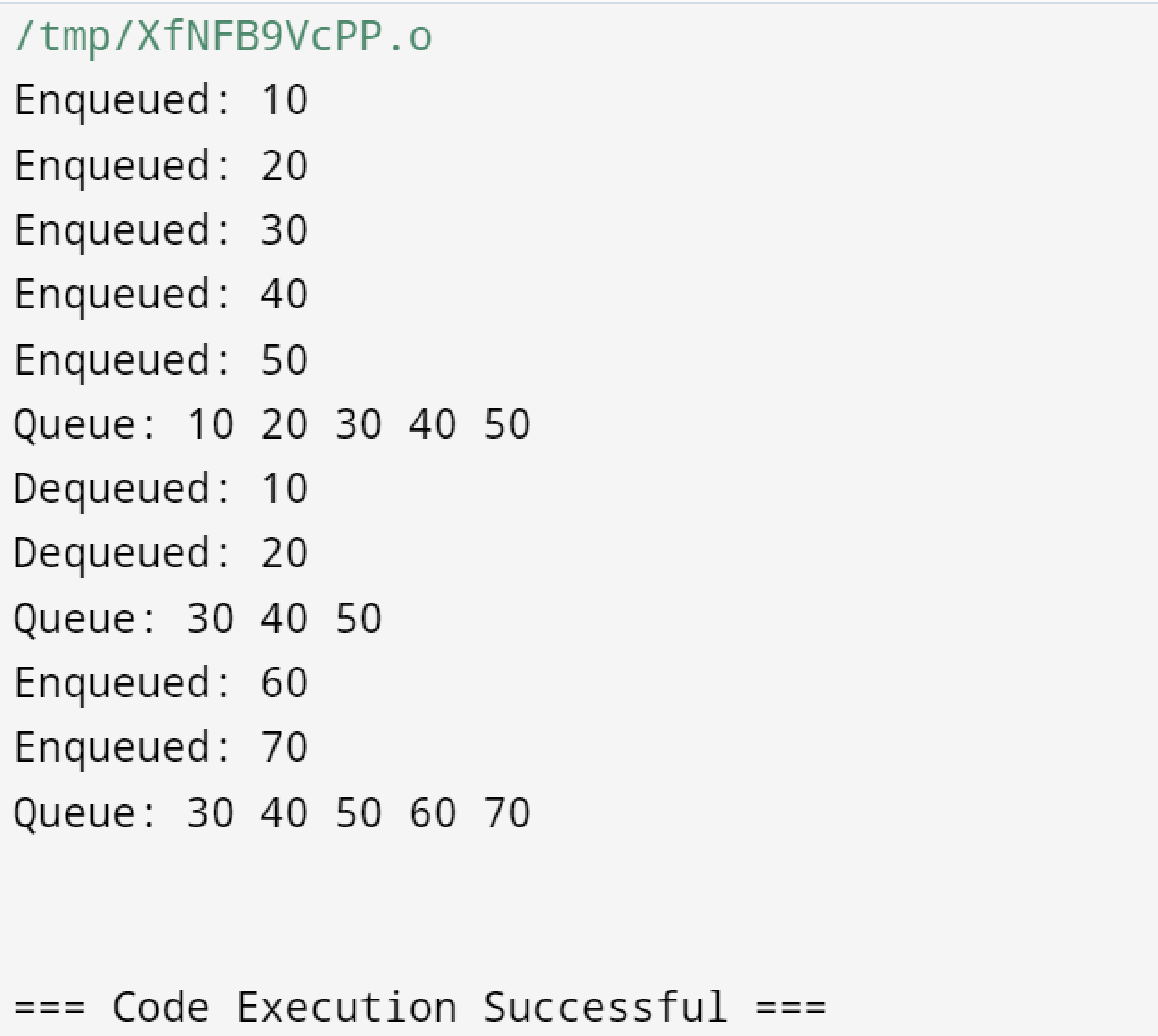
1) Implement Circular Queue with insertion and deletion operations.

```
#include <stdio.h>
#define MAX_SIZE 5
int items[MAX_SIZE];
int front = 0, rear = -1, size = 0;
void initQueue()
{
   front = 0;
   rear = -1;
   size = 0;
}
int isFull()
{
   return size == MAX_SIZE;
}
int isEmpty()
{
   return size == 0;
}
void enqueue(int item)
{
   if (isFull())
   {
      printf("Queue is full\n");
      return;
   }
   rear = (rear + 1) % MAX_SIZE;
   items[rear] = item;
   size++;
   printf("Enqueued: %d\n", item);
}
```

```
int dequeue()
{
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return -1;
    }
    int item = items[front];
    front = (front + 1) % MAX_SIZE;
    size--;
    printf("Dequeued: %d\n", item);
    return item;
}
int peek()
{
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return -1;
    }
    return items[front];
}
void display()
{
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue: ");
    int i;
    for (i = 0; i < size; i++)
    {
        printf("%d ", items[(front + i) % MAX_SIZE]);
    }
    printf("\n");
}
int main()
{
    initQueue();
    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);
    enqueue(50);
    display();
    dequeue();
    dequeue();
    display();
    enqueue(60);
    enqueue(70);
    display();
    return 0;
}
```

OUT PUT:



```
/tmp/XfNFB9VcPP.o
Enqueued: 10
Enqueued: 20
Enqueued: 30
Enqueued: 40
Enqueued: 50
Queue: 10 20 30 40 50
Dequeued: 10
Dequeued: 20
Queue: 30 40 50
Enqueued: 60
Enqueued: 70
Queue: 30 40 50 60 70


=== Code Execution Successful ===
```

2) Implement Double Ended Queue with insertion and deletion operations

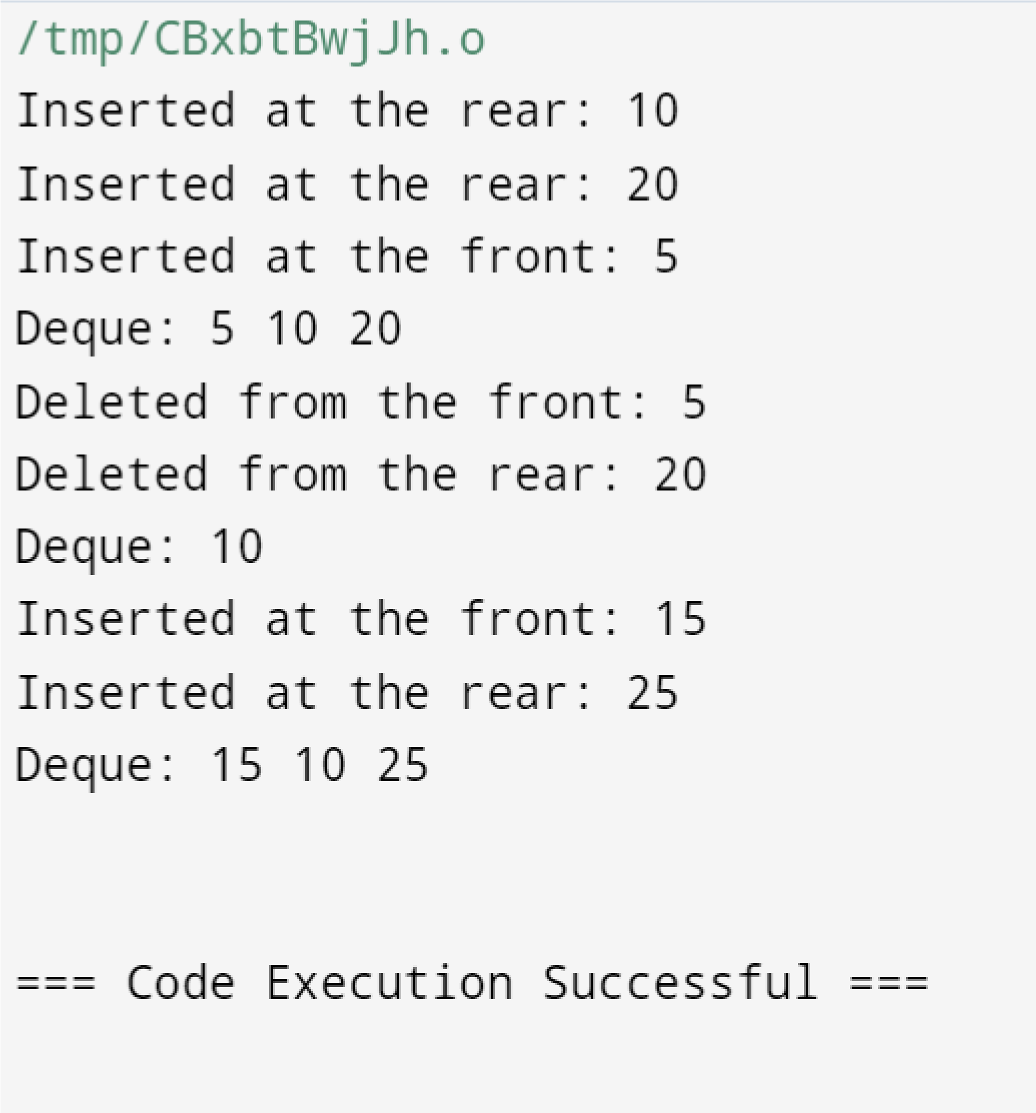
```
#include <stdio.h>
#define MAX_SIZE 5
int items[MAX_SIZE];
int front = 0, rear = -1, size = 0;
void initDeque()
{
   front = 0;
   rear = -1;
   size = 0;
}
int isFull()
```

```
{
    return size == MAX_SIZE;
}
int isEmpty()
{
    return size == 0;
}
void insertFront(int item)
{
    if (isFull())
    {
        printf("Deque is full\n");
        return;
    }
    front = (front - 1 + MAX_SIZE) % MAX_SIZE;
    items[front] = item;
    size++;
    printf("Inserted at the front: %d\n", item);
}
void insertRear(int item)
{
    if (isFull())
    {
        printf("Deque is full\n");
        return;
    }
    rear = (rear + 1) % MAX_SIZE;
    items[rear] = item;
    size++;
    printf("Inserted at the rear: %d\n", item);
}
```

```
int deleteFront()
{
  if (isEmpty())
  {
    printf("Deque is empty\n");
    return -1;
  }
  int item = items[front];
  front = (front + 1) % MAX_SIZE;
  size--;
  printf("Deleted from the front: %d\n", item);
  return item;
}
int deleteRear()
{
  if (isEmpty())
  {
    printf("Deque is empty\n");
    return -1;
  }
  int item = items[rear];
  rear = (rear - 1 + MAX_SIZE) % MAX_SIZE;
  size--;
  printf("Deleted from the rear: %d\n", item);
  return item;
}
void display()
{
  if (isEmpty())
  {
    printf("Deque is empty\n");
```

```
        return;
    }
    printf("Deque: ");
    int i;
    for (i = 0; i < size; i++)
    {
        printf("%d ", items[(front + i) % MAX_SIZE]);
    }
    printf("\n");
}
int main()
{
    initDeque();
    insertRear(10);
    insertRear(20);
    insertFront(5);
    display();
    deleteFront();
    deleteRear();
    display();
    insertFront(15);
    insertRear(25);
    display();
    return 0;
}
```

OUT PUT:



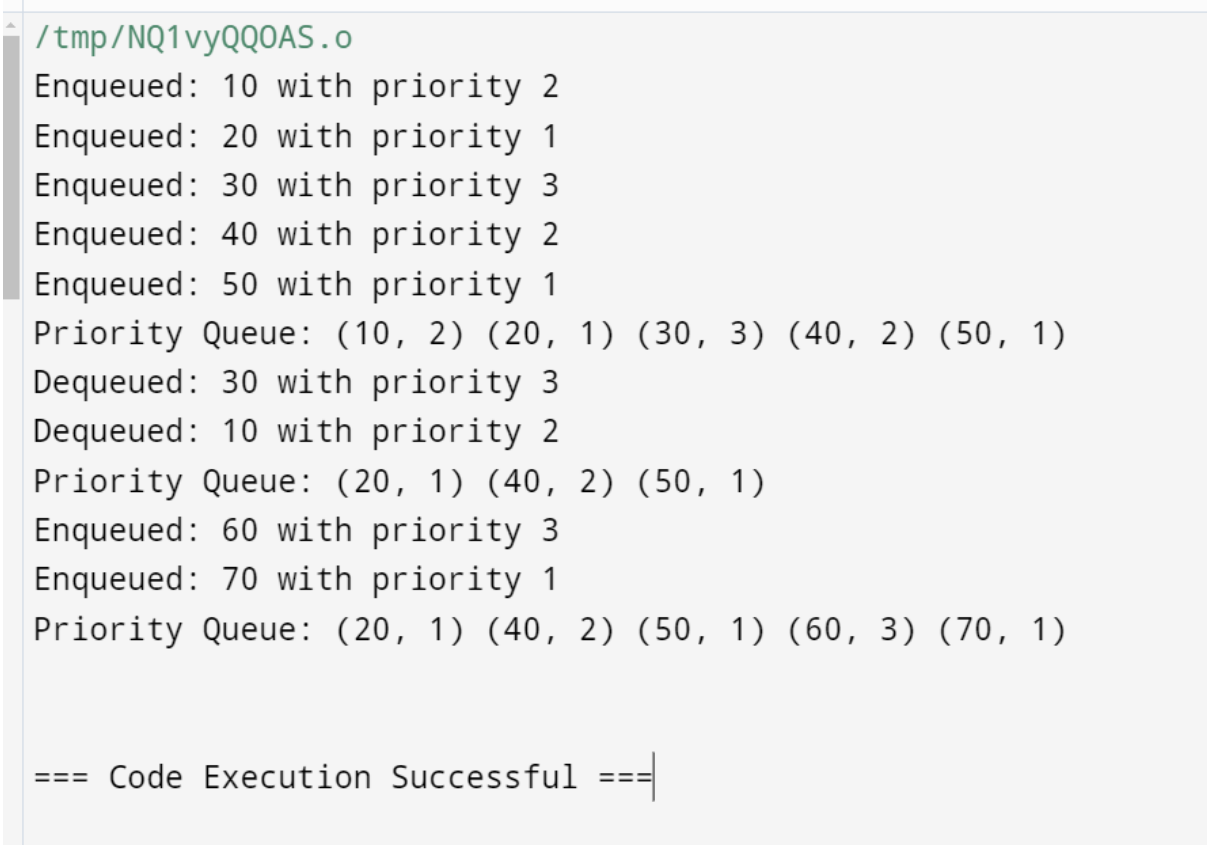
```
/tmp/CBxbtBwjJh.o
Inserted at the rear: 10
Inserted at the rear: 20
Inserted at the front: 5
Deque: 5 10 20
Deleted from the front: 5
Deleted from the rear: 20
Deque: 10
Inserted at the front: 15
Inserted at the rear: 25
Deque: 15 10 25


=== Code Execution Successful ===
```

3)Implement Priority Queue with insertion and deletion operations

```
#include <stdio.h>
#define MAX_SIZE 5
int data[MAX_SIZE];
int priority[MAX_SIZE];
int front = 0, rear = -1, size = 0;
void initQueue()
{
   front = 0;
```

```
    rear = -1;
    size = 0;
}
int isFull()
{
    return size == MAX_SIZE;
}
int isEmpty()
{
    return size == 0;
}
void enqueue(int dataValue, int priorityValue)
{
    if (isFull())
    {
        printf("Queue is full\n");
        return;
    }
    rear = (rear + 1) % MAX_SIZE;
    data[rear] = dataValue;
    priority[rear] = priorityValue;
    size++;
    printf("Enqueued: %d with priority %d\n", dataValue, priorityValue);
}
int dequeue()
{
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return -1;
    }
    int highestPriority = 0;
    int index = 0;
    for (int i = 0; i < size; i++)
    {
        if (priority[i] > highestPriority)
        {
            highestPriority = priority[i];
            index = i;
        }
    }
    int item = data[index];
    for (int i = index; i < size - 1; i++)
    {
        data[i] = data[i + 1];
        priority[i] = priority[i + 1];
    }
    rear = (rear - 1 + MAX_SIZE) % MAX_SIZE;
    size--;
    printf("Dequeued: %d with priority %d\n", item, highestPriority);
    return item;
}
void display()
{
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Priority Queue: ");
    for (int i = 0; i < size; i++)
    {
        printf("(%d, %d) ", data[i], priority[i]);
    }
    printf("\n");
}
int main()
{
    initQueue();
    enqueue(10, 2);
    enqueue(20, 1);
    enqueue(30, 3);
    enqueue(40, 2);
    enqueue(50, 1);
    display();
    dequeue();
    dequeue();
    display();
    enqueue(60, 3);
    enqueue(70, 1);
    display();
    return 0;
}
```

OUT PUT:



```
/tmp/NQ1vyQQOAS.o
Enqueued: 10 with priority 2
Enqueued: 20 with priority 1
Enqueued: 30 with priority 3
Enqueued: 40 with priority 2
Enqueued: 50 with priority 1
Priority Queue: (10, 2) (20, 1) (30, 3) (40, 2) (50, 1)
Dequeued: 30 with priority 3
Dequeued: 10 with priority 2
Priority Queue: (20, 1) (40, 2) (50, 1)
Enqueued: 60 with priority 3
Enqueued: 70 with priority 1
Priority Queue: (20, 1) (40, 2) (50, 1) (60, 3) (70, 1)


=== Code Execution Successful ===
```