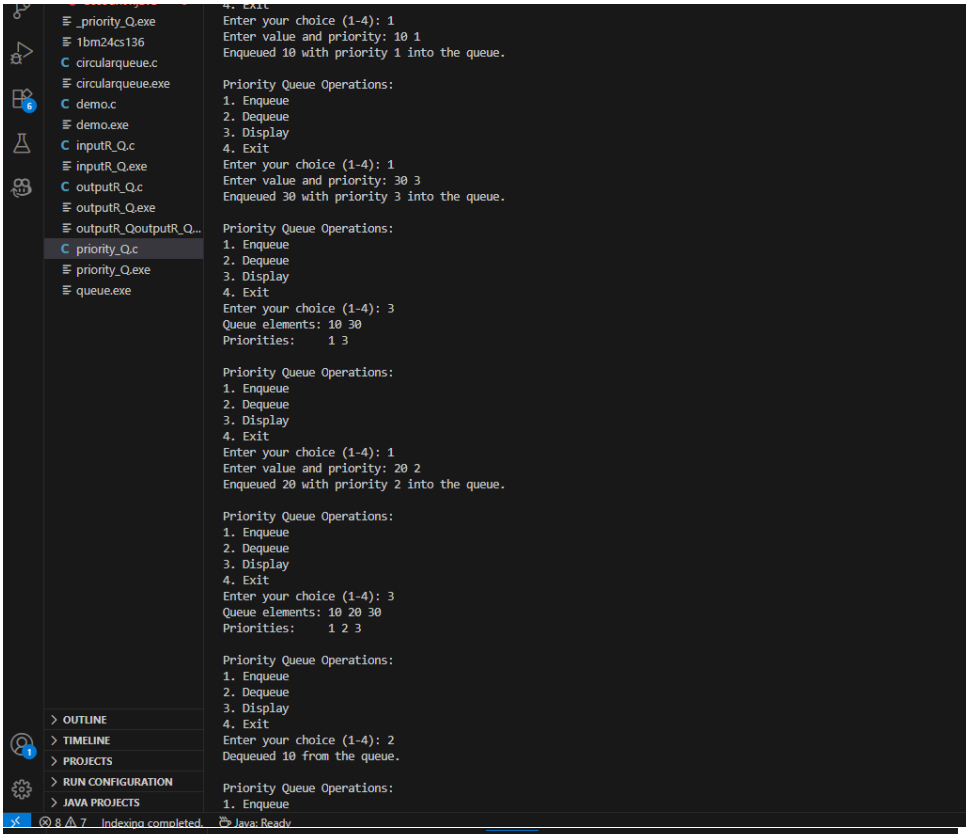
1. Priority Queue:-

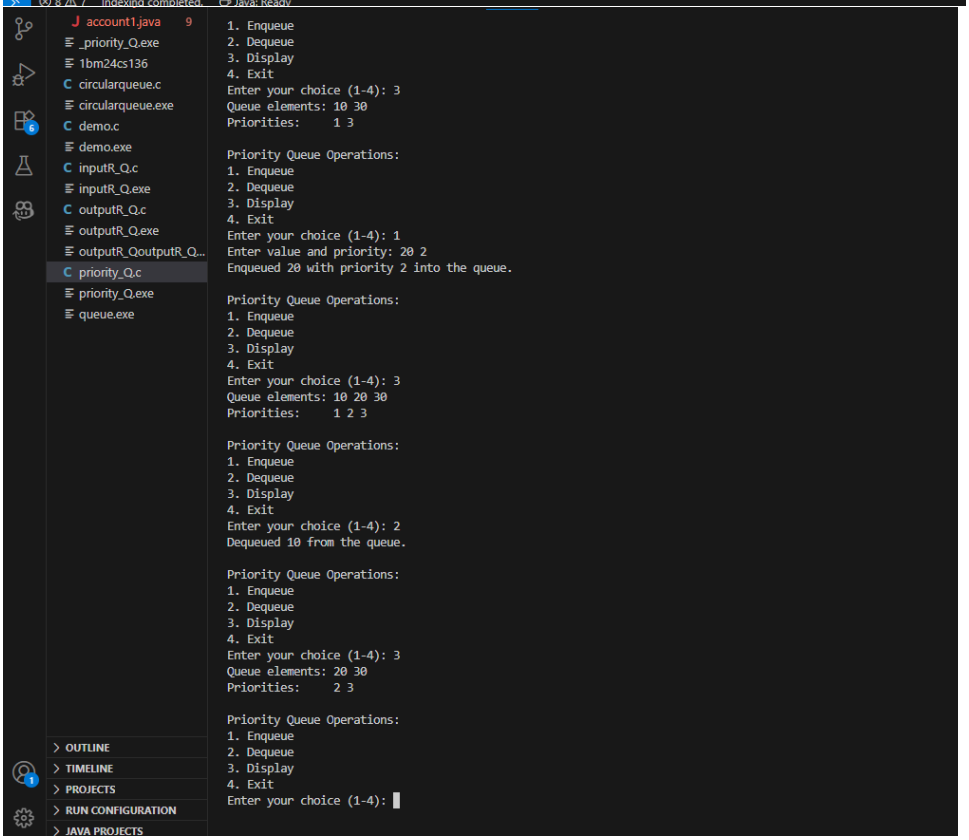
```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int priority[MAX];
int front = -1, rear = -1;

int isFull() {
        return rear == MAX - 1;
    }

    int isEmpty() {
        return front == -1 || front > rear;
    }

    void enqueue(int value, int p) {
        if (isFull()) {
            printf("Queue Overflow! Cannot enqueue %d\n", value);
            return;
        }

        if (front == -1) {
            front = 0;
            rear = 0;
            queue[rear] = value;
            priority[rear] = p;
            printf("Enqueued %d with priority %d into the
  queue.\n", value, p);
            return;
        }

        int i;
        for (i = rear; i >= front && priority[i] > p; i--) {
            queue[i + 1] = queue[i];
            priority[i + 1] = priority[i];
        }

        queue[i + 1] = value;
        priority[i + 1] = p;
        rear++;

```

```
42.           printf("Enqueued %d with priority %d into the queue.\n",
    value, p);
43.       }
44.
45.       void dequeue() {
46.           if (isEmpty()) {
47.               printf("Queue Underflow! Queue is empty.\n");
48.               return;
49.           }
50.
51.           int dequeuedValue = queue[front];
52.           front++;
53.
54.           if (front > rear) {
55.               front = -1;
56.               rear = -1;
57.           }
58.
59.           printf("Dequeued %d from the queue.\n", dequeuedValue);
60.       }
61.
62.       void display() {
63.           if (isEmpty()) {
64.               printf("Queue is empty.\n");
65.               return;
66.           }
67.
68.           printf("Queue elements: ");
69.           for (int i = front; i <= rear; i++)
70.               printf("%d ", queue[i]);
71.
72.           printf("\nPriorities:     ");
73.           for (int i = front; i <= rear; i++)
74.               printf("%d ", priority[i]);
75.
76.           printf("\n");
77.       }
78.
79.       int main() {
80.           int choice, value, p;
81.
82.           while (1) {
```

```
83.             printf("\nPriority Queue Operations:\n");
84.             printf("1. Enqueue\n");
85.             printf("2. Dequeue\n");
86.             printf("3. Display\n");
87.             printf("4. Exit\n");
88.             printf("Enter your choice (1-4): ");
89.             scanf("%d", &choice);
90.
91.             switch (choice) {
92.                 case 1:
93.                     printf("Enter value and priority: ");
94.                     scanf("%d %d", &value, &p);
95.                     enqueue(value, p);
96.                     break;
97.                 case 2:
98.                     dequeue();
99.                     break;
100.                case 3:
101.                    display();
102.                    break;
103.                case 4:
104.                    printf("Exiting program.\n");
105.                    return 0;
106.                default:
107.                    printf("Invalid choice! Please enter 1-4.\n");
108.            }
109.        }
110.
111.        return 0;
112.    }
```

OUTPUT:-


_priority_Q.exe
1bm24cs136
circularqueue.c
circularqueue.exe
demo.c
demo.exe
inputR_Q.c
inputR_Q.exe
outputR_Q.c
outputR_Q.exe
outputR_QoutputR_Q...
priority_Q.c
priority_Q.exe
queue.exe
OUTLINE
TIMELINE
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
Indexing completed.
Java: Ready
4. Exit
Enter your choice (1-4): 1
Enter value and priority: 10 1
Enqueued 10 with priority 1 into the queue.
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 1
Enter value and priority: 30 3
Enqueued 30 with priority 3 into the queue.
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 10 30
Priorities: 1 3
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 1
Enter value and priority: 20 2
Enqueued 20 with priority 2 into the queue.
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 10 20 30
Priorities: 1 2 3
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 2
Dequeued 10 from the queue.
Priority Queue Operations:
1. Enqueue


account1.java 9
_priority_Q.exe
1bm24cs136
circularqueue.c
circularqueue.exe
demo.c
demo.exe
inputR_Q.c
inputR_Q.exe
outputR_Q.c
outputR_Q.exe
outputR_QoutputR_Q...
priority_Q.c
priority_Q.exe
queue.exe
OUTLINE
TIMELINE
PROJECTS
RUN CONFIGURATION
JAVA PROJECTS
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 10 30
Priorities: 1 3
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 1
Enter value and priority: 20 2
Enqueued 20 with priority 2 into the queue.
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 10 20 30
Priorities: 1 2 3
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 2
Dequeued 10 from the queue.
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4): 3
Queue elements: 20 30
Priorities: 2 3
Priority Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1-4):

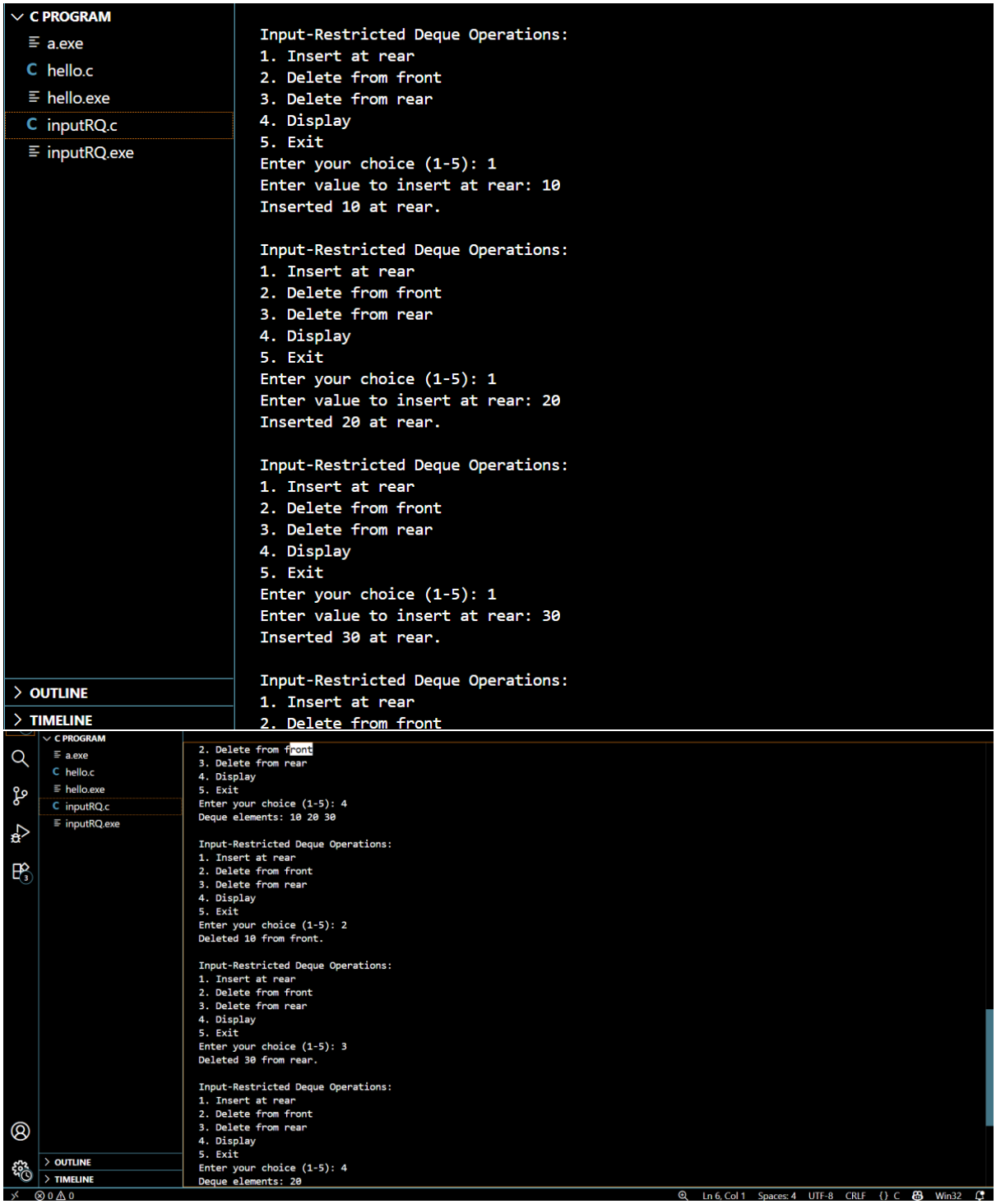
2.Input restricted Queue:-

```
#include <stdio.h>
#define MAX 5

int deque[MAX];
int front = -1, rear = -1;


int isFull() {
   return rear == MAX - 1;
}


int isEmpty() {
   return front == -1 || front > rear;
}

void insertRear(int value) {
   if (isFull()) {
      printf("Deque Overflow! Cannot insert %d\n", value);
      return;
   }
   if (front == -1) {
      front = 0;
      rear = 0;
   } else {
      rear++;
   }
   deque[rear] = value;
   printf("Inserted %d at rear.\n", value);
}


void deleteFront() {
   if (isEmpty()) {
      printf("Deque Underflow! Deque is empty.\n");
      return;
   }
   int deleted = deque[front];
   front++;
   if (front > rear) {
      front = -1;
      rear = -1;
   }
   printf("Deleted %d from front.\n", deleted);
}


void deleteRear() {
   if (isEmpty()) {
      printf("Deque Underflow! Deque is empty.\n");
      return;
```

```
    }
    int deleted = deque[rear];
    rear--;
    if (front > rear) {
        front = -1;
        rear = -1;
    }
    printf("Deleted %d from rear.\n", deleted);
}

void display() {
    if (isEmpty()) {
        printf("Deque is empty.\n");
        return;
    }
    printf("Deque elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", deque[i]);
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\nInput-Restricted Deque Operations:\n");
        printf("1. Insert at rear\n");
        printf("2. Delete from front\n");
        printf("3. Delete from rear\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice (1-5): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at rear: ");
                scanf("%d", &value);
                insertRear(value);
                break;
            case 2:
                deleteFront();
                break;
            case 3:
                deleteRear();
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exiting program.\n");
                return 0;
```

```
        default:
            printf("Invalid choice! Enter 1-5.\n");
      }
   }
   return 0;
}
```

OUTPUT:-



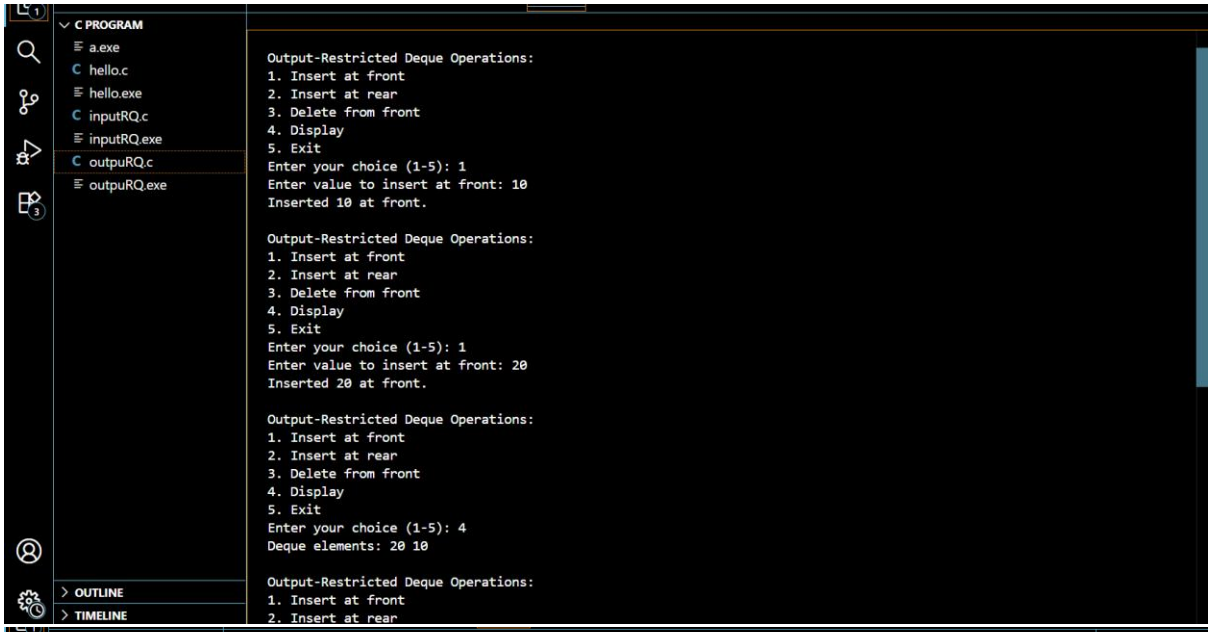
*Output Restricted Queue:-

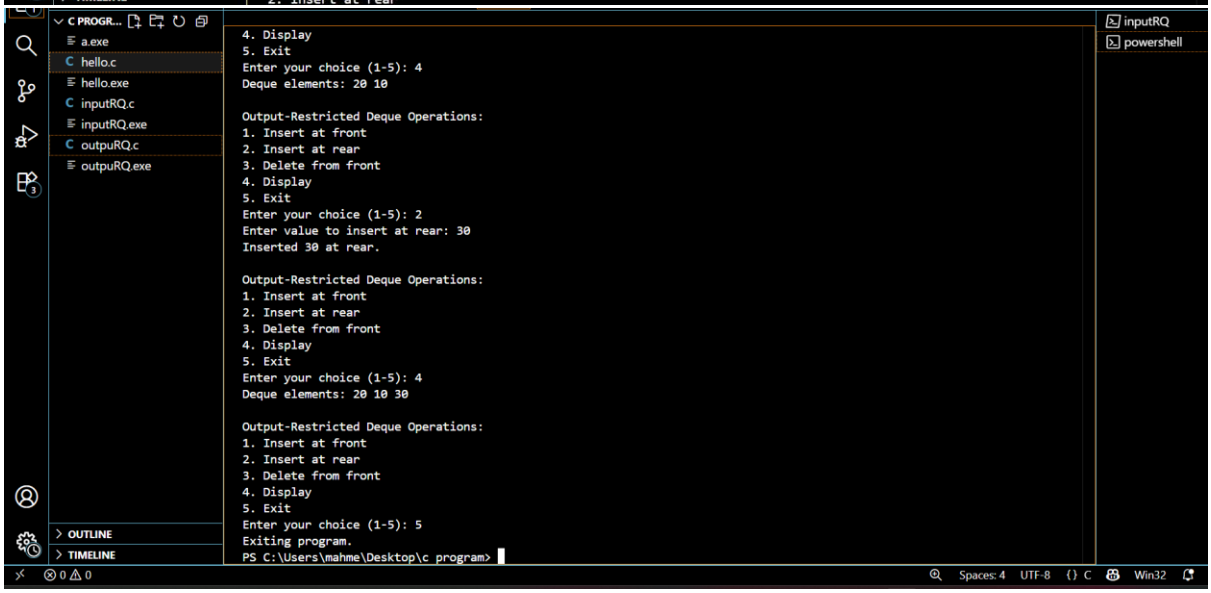
```
#include <stdio.h>
#define MAX 5

int deque[MAX];
int front = -1, rear = -1;

int isFull() {
   return rear == MAX - 1;
}

int isEmpty() {
   return front == -1 || front > rear;
}

void insertFront(int value) {
   if (isFull()) {
      printf("Deque Overflow! Cannot insert %d\n", value);
      return;
   }
   if (front == -1) {
      front = 0;
      rear = 0;
   } else {

      for (int i = rear; i >= front; i--) {
         deque[i + 1] = deque[i];
      }
      rear++;
   }
   deque[front] = value;
   printf("Inserted %d at front.\n", value);
}

void insertRear(int value) {
   if (isFull()) {
      printf("Deque Overflow! Cannot insert %d\n", value);
      return;
   }
   if (front == -1) {
      front = 0;
      rear = 0;
   } else {
      rear++;
   }
   deque[rear] = value;
   printf("Inserted %d at rear.\n", value);
}

void deleteFront() {
   if (isEmpty()) {
      printf("Deque Underflow! Deque is empty.\n");
      return;
```

```
    }
    int deleted = deque[front];
    front++;
    if (front > rear) {
        front = -1;
        rear = -1;
    }
    printf("Deleted %d from front.\n", deleted);
}

void display() {
    if (isEmpty()) {
        printf("Deque is empty.\n");
        return;
    }
    printf("Deque elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", deque[i]);
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\nOutput-Restricted Deque Operations:\n");
        printf("1. Insert at front\n");
        printf("2. Insert at rear\n");
        printf("3. Delete from front\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice (1-5): ");
        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:
                deleteFront();
                break;
            case 4:
                display();
                break;
            case 5:
```

```
            printf("Exiting program.\n");
            return 0;
        default:
            printf("Invalid choice! Enter 1-5.\n");
    }
  }
  return 0;
}
```

OUTPUT:-



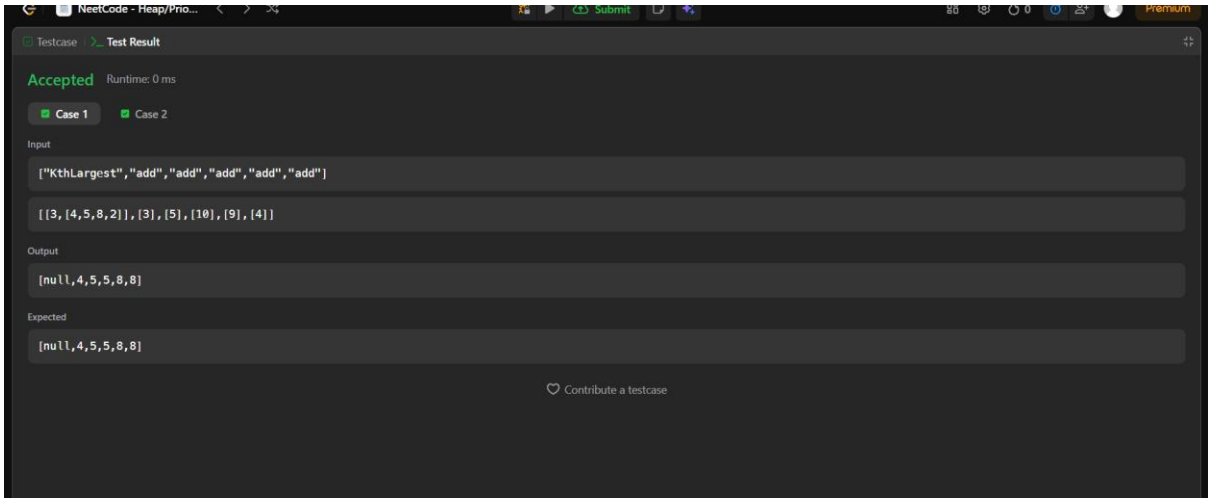


LEETCODE:-

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

typedef struct {
    int k;
```

```
    int size;
    int arr[1000];
} KthLargest;


void sortDescending(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] < arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}


KthLargest* kthLargestCreate(int k, int* nums, int numsSize) {
    KthLargest* obj = malloc(sizeof(KthLargest));
    obj->k = k;
    obj->size = numsSize;
    for (int i = 0; i < numsSize; i++) {
        obj->arr[i] = nums[i];
    }
    sortDescending(obj->arr, obj->size);
    return obj;
}

int kthLargestAdd(KthLargest* obj, int val) {
    obj->arr[obj->size++] = val;
    sortDescending(obj->arr, obj->size);
    if (obj->size < obj->k)
        return obj->arr[obj->size - 1];
    return obj->arr[obj->k - 1];
}

void kthLargestFree(KthLargest* obj) {
    free(obj);
}
```

OUTPUT:-

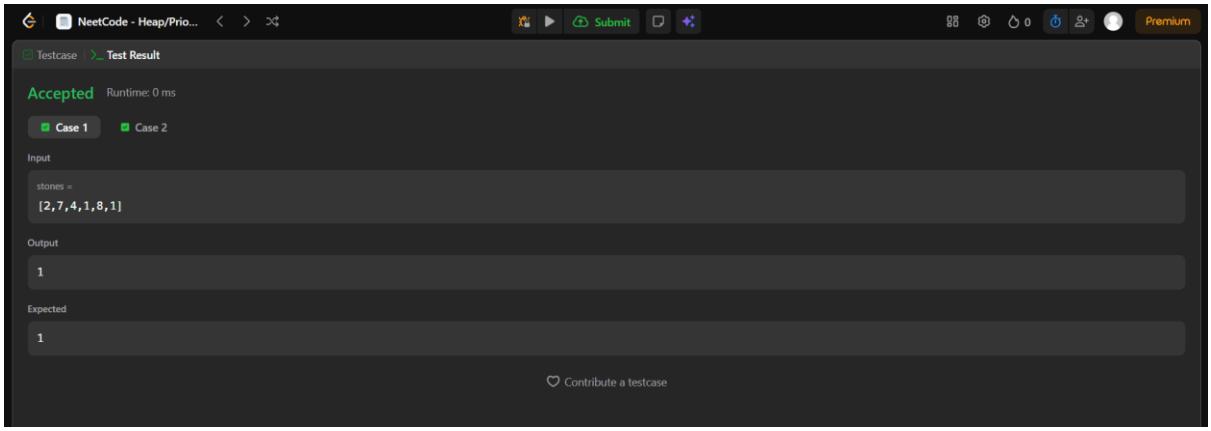


Leedcode:-

```
#include <stdio.h>
void sortDescending(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (arr[i] < arr[j]) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
         }
      }
   }
}

int lastStoneWeight(int stones[], int stonesSize) {
   while (stonesSize > 1) {
      sortDescending(stones, stonesSize);
      int y = stones[0];
      int x = stones[1];

      if (x == y) {

         for (int i = 2; i < stonesSize; i++) {
            stones[i - 2] = stones[i];
         }
         stonesSize -= 2;
      } else {

         stones[0] = y - x;
         for (int i = 2; i < stonesSize; i++) {
            stones[i - 1] = stones[i];
         }
         stonesSize -= 1;
      }
```

```
    }

    return stonesSize == 0 ? 0 : stones[0];
}
```

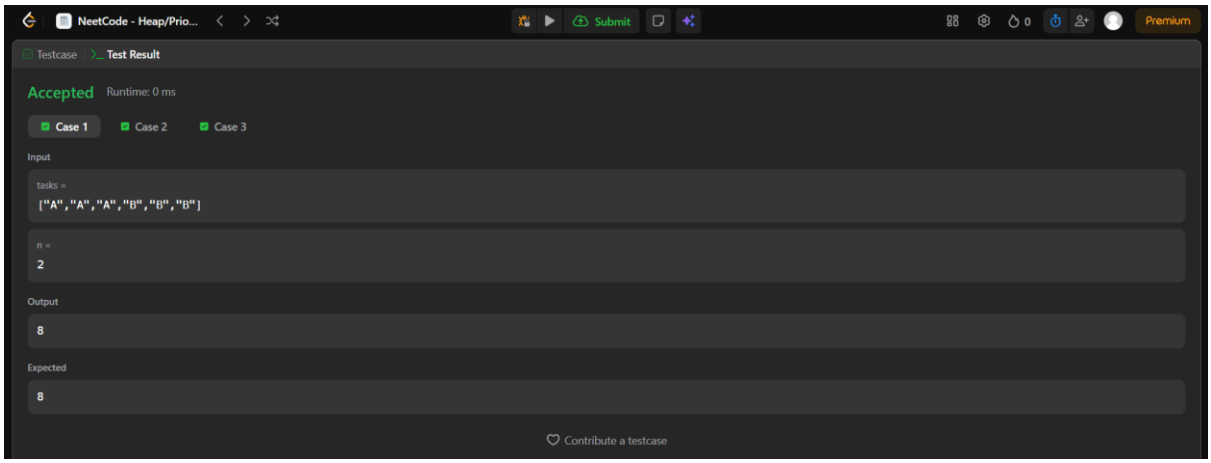
OUTPUT:-



Leetcode:-

```
#include <stdio.h>

int leastInterval(char* tasks, int tasksSize, int n) {
    int freq[26] = {0};
    for (int i = 0; i < tasksSize; i++) {
        freq[tasks[i] - 'A']++;
    }
    int maxFreq = 0;
    for (int i = 0; i < 26; i++) {
        if (freq[i] > maxFreq)
            maxFreq = freq[i];
    }

    int maxCount = 0;
    for (int i = 0; i < 26; i++) {
        if (freq[i] == maxFreq)
            maxCount++;
    }

    int result = (maxFreq - 1) * (n + 1) + maxCount;


    if (result < tasksSize)
        result = tasksSize;

    return result;
}
```

OUTPUT:-


NeetCode - Heap/Prio...
Submit
Premium
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
tasks =
["A","A","A","B","B","B"]
n =
2
Output
8
Expected
8
Contribute a testcase