Experiment 6

CODE:

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

struct Node {
   int customerID;
   struct Node* next;
};

struct Queue {
   struct Node *front, *rear;
};

struct Node* createNode(int customerID) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->customerID = customerID;
   newNode->next = NULL;
   return newNode;
}

struct Queue* createQueue() {
   struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
   q->front = q->rear = NULL;
   return q;
}

void enqueue(struct Queue* q, int customerID) {
   struct Node* newNode = createNode(customerID);

   if (q->rear == NULL) {
      q->front = q->rear = newNode;
   } else {
      q->rear->next = newNode;
      q->rear = newNode;
```

```
    }

    printf("Customer %d added to the queue.\n", customerID);
}

void dequeue(struct Queue* q) {
    if (q->front == NULL) {
        printf("Queue is empty. No customers to serve.\n");
        return;
    }

    struct Node* temp = q->front;
    q->front = q->front->next;

    if (q->front == NULL) {
        q->rear = NULL;
    }

    printf("Served customer with ID: %d\n", temp->customerID);
    free(temp);
}

void insertVIP(struct Queue* q, int customerID) {
    struct Node* newNode = createNode(customerID);

    if (q->front == NULL) {

        q->front = q->rear = newNode;
    } else {
        struct Node* slow = q->front;
        struct Node* fast = q->front;
        struct Node* prev = NULL;

        while (fast != NULL && fast->next != NULL) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        }

        newNode->next = slow;
        if (prev != NULL) {
            prev->next = newNode;
        } else {
            q->front = newNode;
        }

        printf("VIP customer %d added to the middle of the queue.\n",
customerID);
    }
}

void insertVVIP(struct Queue* q, int customerID) {
    struct Node* newNode = createNode(customerID);

    if (q->front == NULL) {
        q->front = q->rear = newNode;
    } else {
        newNode->next = q->front;
        q->front = newNode;
    }
    printf("VVIP customer %d added to the front of the queue.\n",
customerID);
}
int isQueueEmpty(struct Queue* q) {
    return q->front == NULL;
}
void displayQueue(struct Queue* q) {
    struct Node* temp = q->front;
    if (temp == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Current queue: ");
    while (temp != NULL) {
        printf("%d ", temp->customerID);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Queue* q = createQueue();
    int choice, customerID;

    while (1) {
        printf("\nQueue Operations Menu:\n");
        printf("1. Add Customer to Queue (Enqueue)\n");
        printf("2. Serve Customer (Dequeue)\n");
        printf("3. Display Queue\n");
        printf("4. Insert VIP Customer (Middle of Queue)\n");
        printf("5. Insert VVIP Customer (Front of Queue)\n");
        printf("6. Check if Queue is Empty\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter customer ID to add to the queue: ");
                scanf("%d", &customerID);
                enqueue(q, customerID);
                break;
            case 2:
                dequeue(q);
                break;
            case 3:
                displayQueue(q);
                break;
            case 4:
                printf("Enter VIP customer ID to insert in the middle of the
queue: ");
                scanf("%d", &customerID);
                insertVIP(q, customerID);
                break;
            case 5:
                printf("Enter VVIP customer ID to insert at the front of the queue:
");
                scanf("%d", &customerID);
                insertVVIP(q, customerID);
                break;
            case 6:
                if (isQueueEmpty(q)) {
                  printf("Queue is empty.\n");
                } else {
                  printf("Queue is not empty.\n");
                }
                break;
            case 7:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
```

```
7. Exit
Enter your choice: 1
Enter customer ID to add to the queue: 101
Customer 101 added to the queue.

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 1
Enter customer ID to add to the queue: 102
Customer 102 added to the queue.

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 1
Enter customer ID to add to the queue: 103
Customer 103 added to the queue.

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 1
Enter customer ID to add to the queue: 104
Customer 104 added to the queue.
```

```
Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 106
Invalid choice! Please try again.

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 2
Served customer with ID: 101

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 3
Current queue: 102 103 104

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
```

```
6. Check if Queue is Empty
7. Exit
Enter your choice: 4
Enter VIP customer ID to insert in the middle of the queue: 555
VIP customer 555 added to the middle of the queue.

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 3
Current queue: 102 555 103 104

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 5
Enter VVIP customer ID to insert at the front of the queue: 999
VVIP customer 999 added to the front of the queue.

Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 3
Current queue: 999 102 555 103 104
```

```
Queue Operations Menu:
1. Add Customer to Queue (Enqueue)
2. Serve Customer (Dequeue)
3. Display Queue
4. Insert VIP Customer (Middle of Queue)
5. Insert VVIP Customer (Front of Queue)
6. Check if Queue is Empty
7. Exit
Enter your choice: 7
Exiting the program.


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