```c
//1.

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

#define BUY 1
#define SELL 2

struct Order {
   int price;
   int quantity;
   int type;
};

struct OrderQueue {
   int size;
   int front;
   int rear;
   struct Order *orders;
};


void createQueue(struct OrderQueue *, int);
void enqueue(struct OrderQueue *, struct Order);
void displayQueue(struct OrderQueue);
struct Order dequeue(struct OrderQueue *);
void matchOrders(struct OrderQueue *, struct OrderQueue *);

int main() {
   struct OrderQueue buyQueue, sellQueue;
```

```
createQueue(&buyQueue, 5);
createQueue(&sellQueue, 5);

struct Order buy1 = {100, 50, BUY};
struct Order buy2 = {105, 30, BUY};
struct Order sell1 = {90, 40, SELL};
struct Order sell2 = {95, 20, SELL};
struct Order sell3 = {105, 10, SELL};

enqueue(&buyQueue, buy1);
enqueue(&buyQueue, buy2);
enqueue(&sellQueue, sell1);
enqueue(&sellQueue, sell2);
enqueue(&sellQueue, sell3);

printf("Buy Orders:\n");
displayQueue(buyQueue);

printf("\nSell Orders:\n");
displayQueue(sellQueue);

printf("\nMatching Orders:\n");
matchOrders(&buyQueue, &sellQueue);

printf("\nBuy Orders After Matching:\n");
displayQueue(buyQueue);

printf("\nSell Orders After Matching:\n");
displayQueue(sellQueue);

return 0;
```

```
}

void createQueue(struct OrderQueue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->orders = (struct Order *)malloc(q->size * sizeof(struct Order));
}

void enqueue(struct OrderQueue *q, struct Order order) {
    if (q->rear == q->size - 1) {
        printf("Queue is full, cannot add more orders.\n");
    } else {
        q->rear++;
        q->orders[q->rear] = order;
    }
}

struct Order dequeue(struct OrderQueue *q) {
    struct Order order = {0, 0, 0};
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        order = q->orders[q->front];
    }
    return order;
}

void displayQueue(struct OrderQueue q) {
    if (q.front == q.rear) {
        printf("Queue is empty.\n");
        return;
    }
    for (int i = q.front + 1; i <= q.rear; i++) {
        struct Order order = q.orders[i];
        printf("Price: %d, Quantity: %d, Type: %s\n", order.price, order.quantity, order.type == BUY ?
"BUY" : "SELL");
    }
}



void matchOrders(struct OrderQueue *buyQueue, struct OrderQueue *sellQueue) {
    while (buyQueue->front != buyQueue->rear && sellQueue->front != sellQueue->rear) {
        struct Order buyOrder = buyQueue->orders[buyQueue->front + 1];
        struct Order sellOrder = sellQueue->orders[sellQueue->front + 1];



        if (buyOrder.price >= sellOrder.price) {
            int matchQuantity = (buyOrder.quantity < sellOrder.quantity) ? buyOrder.quantity :
sellOrder.quantity;

            printf("Matched %d units at price %d\n", matchQuantity, sellOrder.price);



            buyOrder.quantity -= matchQuantity;
            sellOrder.quantity -= matchQuantity;

            dequeue(buyQueue);
            dequeue(sellQueue);

            if (buyOrder.quantity > 0) {
                enqueue(buyQueue, buyOrder);
            }
            if (sellOrder.quantity > 0) {
                enqueue(sellQueue, sellOrder);
            }
        } else {
            break;
        }
    }
}

//2.

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

struct Customer {
    char name[50];
    int priority;
    char timestamp[20];
    struct Customer* next;
} *front = NULL, *rear = NULL;

void enqueue(const char *name, int priority, const char *timestamp);
void display();
void dequeue();
void processQueue();

int main() {
    enqueue("Sofia", 1, "2025-01-21 09:00");
    enqueue("Mickelen", 2, "2025-01-21 09:15");
    enqueue("Christo", 1, "2025-01-21 09:30");
    printf("Customer Queue before processing:\n");
    display();
    processQueue();
    printf("\nCustomer Queue after processing:\n");
    display();
    return 0;
}

void enqueue(const char *name, int priority, const char *timestamp) {
    struct Customer* newCustomer = (struct Customer*)malloc(sizeof(struct Customer));
    if (newCustomer == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    strncpy(newCustomer->name, name, sizeof(newCustomer->name) - 1);
    newCustomer->priority = priority;
    strncpy(newCustomer->timestamp, timestamp, sizeof(newCustomer->timestamp) - 1);
    newCustomer->next = NULL;
    if (front == NULL || front->priority > priority) {
        newCustomer->next = front;
        front = newCustomer;
        if (rear == NULL) rear = newCustomer;
    } else {
        struct Customer* current = front;
        while (current->next != NULL && current->next->priority <= priority)
            current = current->next;
        newCustomer->next = current->next;
        current->next = newCustomer;
        if (newCustomer->next == NULL) rear = newCustomer;
    }
    printf("Enqueued customer: Name = %s, Priority = %d, Timestamp = %s\n", name, priority,
timestamp);
}

void display() {
    struct Customer* current = front;
    if (current == NULL) {
        printf("Queue is empty\n");
        return;
    }
    while (current != NULL) {
        printf("Name = %s, Priority = %d, Timestamp = %s\n", current->name, current->priority, current-
>timestamp);
        current = current->next;
    }
}

void dequeue() {
    if (front == NULL) {
        printf("Queue is empty, no customer to serve\n");
        return;
    }
    struct Customer* temp = front;
    printf("Serving customer: Name = %s, Priority = %d, Timestamp = %s\n", front->name, front-
>priority, front->timestamp);
    front = front->next;
    if (front == NULL) rear = NULL;
    free(temp);
}

void processQueue() {
    while (front != NULL) dequeue();
```

```
}

//3.

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

struct Queue {
   int size;
   int front;
   int rear;
   struct Attendee *Q;
};

struct Attendee {
   char name[100];
   int isVIP;
};

void create(struct Queue *, int);
void enqueue(struct  Queue *, struct Attendee);
struct Attendee dequeue(struct  Queue *);
void display(struct  Queue);

int main() {
   struct Queue vipQueue, regularQueue;

   create(&vipQueue, 5);
   create(&regularQueue, 5);

   struct Attendee attendee1 = {"Sofi", 1};
```

```
    struct Attendee attendee2 = {"Mickey", 0};
    struct Attendee attendee3 = {"Sanjay", 1};

    enqueue(&vipQueue, attendee1);
    enqueue(&regularQueue, attendee2);
    enqueue(&vipQueue, attendee3);

    printf("VIP Queue: \n");
    display(vipQueue);
    printf("Regular Queue: \n");
    display(regularQueue);

    printf("\nChecking in attendees:\n");
    printf("VIP Attendee Checked-in: %s\n", dequeue(&vipQueue).name);
    printf("Regular Attendee Checked-in: %s\n", dequeue(&regularQueue).name);



    printf("\nUpdated VIP Queue: \n");
    display(vipQueue);
    printf("Updated Regular Queue: \n");
    display(regularQueue);

    return 0;
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->Q = (struct Attendee *)malloc(q->size * sizeof(struct Attendee));
}
```

```
void enqueue(struct Queue *q, struct Attendee attendee) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->Q[q->rear] = attendee;
    }
}

struct Attendee dequeue(struct Queue *q) {
    struct Attendee temp;
    if (q->front == q->rear) {
        printf("Queue is empty\n");
        temp.isVIP = -1;
        return temp;
    } else {
        q->front++;
        return q->Q[q->front];
    }
}

void display(struct Queue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
    } else {
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Name: %s, VIP: %d\n", q.Q[i].name, q.Q[i].isVIP);
        }
    }
}
```

```
//4.

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

struct Customer {
    char name[50];
    int transactionTime;
    struct Customer* next;
} *front = NULL, *rear = NULL;

struct Teller {
    int id;
    int availableTime;
};

void enqueue(const char *name, int transactionTime);
void dequeue();
void processTellers(struct Teller tellers[], int numTellers);
void display();

int main() {
    struct Teller tellers[2];
    for (int i = 0; i < 2; i++) {
        tellers[i].id = i + 1;
        tellers[i].availableTime = 0;
    }
    enqueue("Axs", 5);
    enqueue("bcd", 3);
    enqueue("fsr", 2);
    printf("Customer Queue before processing:\n");
    display();
    processTellers(tellers, 2);
    printf("\nCustomer Queue after processing:\n");
    display();
    return 0;
}

void enqueue(const char *name, int transactionTime) {
    struct Customer* newCustomer = (struct Customer*)malloc(sizeof(struct Customer));
    if (newCustomer == NULL) {
        printf("Queue is full\n");
        return;
    }
    strncpy(newCustomer->name, name, sizeof(newCustomer->name) - 1);
    newCustomer->transactionTime = transactionTime;
    newCustomer->next = NULL;
    if (front == NULL) front = rear = newCustomer;
    else {
        rear->next = newCustomer;
        rear = newCustomer;
    }
    printf("Enqueued customer: Name = %s, Transaction Time = %d\n", name, transactionTime);
}

void dequeue() {
    if (front == NULL) {
        printf("Queue is empty, no customer to serve\n");
        return;
    }
    struct Customer* temp = front;
```

```
  printf("Serving customer: Name = %s, Transaction Time = %d\n", front->name, front-
>transactionTime);
  front = front->next;
  if (front == NULL) rear = NULL;
  free(temp);
}

void processTellers(struct Teller tellers[], int numTellers) {
  while (front != NULL) {
    for (int i = 0; i < numTellers; i++) {
      if (tellers[i].availableTime == 0 && front != NULL) {
        tellers[i].availableTime = front->transactionTime;
        dequeue();
      }
    }
  }
}

void display() {
  struct Customer* current = front;
  if (current == NULL) {
    printf("Queue is empty\n");
    return;
  }
  while (current != NULL) {
    printf("Name = %s, Transaction Time = %d\n", current->name, current->transactionTime);
    current = current->next;
  }
}

//5.
```

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct DataFeed {

    char instrumentName[50];

    float price;

    int timestamp;

};


struct Queue {

    int size;

    int front;

    int rear;

    struct DataFeed *data;

};


void create(struct Queue *, int);

void enqueue(struct  Queue *, struct DataFeed);

void display(struct  Queue);

struct DataFeed dequeue(struct  Queue *);


int main() {

    struct Queue q;

    create(&q, 5);


    struct DataFeed feed1 = {"Instrument1",  100.5f, 1600000001};

    struct DataFeed feed2 = {"Instrument2",  101.0f, 1600000002};

    struct DataFeed feed3 = {"Instrument3",  102.0f, 1600000003};


    enqueue(&q,  feed1);
    enqueue(&q,  feed2);
    enqueue(&q,  feed3);

    display(q);

    struct DataFeed dequeuedData  = dequeue(&q);
    printf("Dequeued:  Instrument: %s, Price: %.2f, Timestamp: %d\n", 
dequeuedData.instrumentName,  dequeuedData.price,  dequeuedData.timestamp);

    display(q);

    return 0;
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->data = (struct DataFeed *)malloc(q->size * sizeof(struct DataFeed));
}

void enqueue(struct Queue *q, struct DataFeed newData) {
    if (q->rear == q->size - 1) {
        printf("Queue  is full, cannot enqueue data.\n");
    } else {
        q->rear++;
        q->data[q->rear] = newData;
        printf("Enqueued:  %s, %.2f, %d\n", newData.instrumentName,  newData.price, 
newData.timestamp);
    }
}

void display(struct Queue q) {
    if (q.front == q.rear) {
        printf("Queue is empty.\n");
    } else {
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Instrument: %s, Price: %.2f, Timestamp: %d\n", q.data[i].instrumentName,
q.data[i].price, q.data[i].timestamp);
        }
    }
    printf("\n");
}

struct DataFeed dequeue(struct Queue *q) {
    struct DataFeed emptyData = {"", 0.0f, 0};
    if (q->front == q->rear) {
        printf("Queue is empty, cannot dequeue data.\n");
    } else {
        q->front++;
        return q->data[q->front];
    }
    return emptyData;
}

//6.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

struct Car
{
    int id;
    struct Car* next;
};

struct Queue
{
    struct Car* front;
    struct Car* rear;
};

void initializeQueue(struct  Queue* q);
int isQueueEmpty(struct  Queue*  q);
void enqueue(struct  Queue*  q, int carId);
void dequeue(struct  Queue*  q);
void trafficLightSystem(struct  Queue*  q);

int main()
{
    struct Queue carQueue;
    initializeQueue(&carQueue);
    enqueue(&carQueue,  1);
    enqueue(&carQueue,  2);
    enqueue(&carQueue,  3);
    trafficLightSystem(&carQueue);
    while (!isQueueEmpty(&carQueue))
        dequeue(&carQueue);
    return 0;
}

void initializeQueue(struct  Queue*  q)
{
    q->front = NULL;
    q->rear = NULL;
}

int isQueueEmpty(struct  Queue*  q)
{
    return q->front == NULL;
}

void enqueue(struct  Queue*  q, int carId)
{
    struct Car* newCar = (struct Car*)malloc(sizeof(struct  Car));
    if (!newCar)
    {
        printf("Memory  allocation  failed\n");
        return;
    }
    newCar->id = carId;
    newCar->next = NULL;
    if (q->rear == NULL)
        q->front  = q->rear = newCar;
    else
    {
        q->rear->next = newCar;
        q->rear = newCar;
    }
    printf("Car %d arrived at the traffic light\n",  carId);
}

void dequeue(struct  Queue*  q)
{
    if (isQueueEmpty(q))
    {
        printf("No cars at the traffic light\n");
        return;
    }
    struct Car* temp = q->front;
    printf("Car %d is passing through the green light.\n", temp->id);
    q->front = q->front->next;
    if (q->front == NULL)
        q->rear = NULL;
    free(temp);
}

void trafficLightSystem(struct Queue* q)
{
    char light = 'R';
    int time = 0;

    while (time < 5)
    {
        printf("\nTime: %d seconds\n", time);
        if (light == 'R')
            printf("Traffic light: RED\n");
        else
        {
            printf("Traffic light: GREEN\n");
            if (!isQueueEmpty(q))
                dequeue(q);
            else
                printf("No cars waiting\n");
        }
        time++;
        sleep(1);
        light = (light == 'R') ? 'G' : 'R';
    }
}

//7.

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

struct Vote {
    char pollingStation[50];
    int voteCount;
    int timestamp;
};

struct Queue {
    int size;
    int front;
    int rear;
    struct Vote *votes;
};

void create(struct Queue *, int);
void enqueue(struct Queue *, struct Vote);
void display(struct Queue);
struct Vote dequeue(struct Queue *);

int main() {
    struct Queue q;
    create(&q, 5);

    struct Vote vote1 = {"Polling Station 1", 100, 1600000001};
    struct Vote vote2 = {"Polling Station 2", 150, 1600000002};
    struct Vote vote3 = {"Polling Station 3", 200, 1600000003};

    enqueue(&q, vote1);
    enqueue(&q, vote2);
    enqueue(&q, vote3);

    display(q);

    struct Vote dequeuedVote = dequeue(&q);
    printf("Dequeued: Polling Station: %s, Votes: %d, Timestamp: %d\n", dequeuedVote.pollingStation,
dequeuedVote.voteCount, dequeuedVote.timestamp);

    display(q);

    return 0;
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->votes = (struct Vote *)malloc(q->size * sizeof(struct Vote));
}

void enqueue(struct Queue *q, struct Vote newVote) {
    if (q->rear == q->size - 1) {
        printf("Queue is full, cannot enqueue vote.\n");
    } else {
        q->rear++;
        q->votes[q->rear] = newVote;
        printf("Vote from %s with %d votes enqueued.\n", newVote.pollingStation, newVote.voteCount);
    }
}

void display(struct Queue q) {
    if (q.front == q.rear) {
        printf("Queue is empty.\n");
    } else {
        printf("Votes from polling stations (from front to rear):\n");
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Polling Station: %s | Votes: %d | Timestamp: %d\n", q.votes[i].pollingStation,
q.votes[i].voteCount, q.votes[i].timestamp);
        }
        printf("\n");
    }
}

struct Vote dequeue(struct Queue *q) {
    struct Vote emptyVote = {"", 0, 0};
    if (q->front == q->rear) {
        printf("Queue is empty, cannot dequeue vote.\n");
    } else {
        q->front++;
        return q->votes[q->front];
    }
    return emptyVote;
}
```

```
//8.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Airplane {
    int id;
    char type[10];
    int priority;
    struct Airplane* next;
};

struct Queue {
    struct Airplane* front;
    struct Airplane* rear;
};

void initializeQueue(struct Queue* q);
int isQueueEmpty(struct Queue* q);
void enqueue(struct Queue* q, int id, const char* type, int priority);
void dequeue(struct Queue* q);
void manageRunway(struct Queue* q, int timeSlots);
void displayQueue(struct Queue* q);

int main() {
    struct Queue runwayQueue;
    initializeQueue(&runwayQueue);

    enqueue(&runwayQueue, 101, "land", 0);
    enqueue(&runwayQueue, 102, "takeoff", 1);
    enqueue(&runwayQueue, 103, "land", 1);
    enqueue(&runwayQueue, 104, "takeoff", 0);
    enqueue(&runwayQueue, 105, "land", 0);

    manageRunway(&runwayQueue, 6);  // 6 time slots to simulate
    return 0;
}

void initializeQueue(struct Queue* q) {
    q->front = NULL;
    q->rear = NULL;
}

int isQueueEmpty(struct Queue* q) {
    return q->front == NULL;
}

void enqueue(struct Queue* q, int id, const char* type, int priority) {
    struct Airplane* newPlane = (struct Airplane*)malloc(sizeof(struct Airplane));
    if (!newPlane) {
        printf("Memory allocation failed!\n");
        return;
    }
    newPlane->id = id;
    strcpy(newPlane->type, type);
    newPlane->priority = priority;
    newPlane->next = NULL;

    if (q->rear == NULL || priority == 1) {
        newPlane->next = q->front;
        q->front = newPlane;
        if (q->rear == NULL) {
            q->rear = newPlane;
        }
    } else {
        q->rear->next = newPlane;
        q->rear = newPlane;
    }
    printf("Airplane %d (%s, priority %d) added to the queue.\n", id, type, priority);
}

void dequeue(struct Queue* q) {
    if (isQueueEmpty(q)) {
        printf("No airplanes in the queue\n");
        return;
    }
    struct Airplane* temp = q->front;
    printf("Airplane %d (%s) is using the runway.\n", temp->id, temp->type);
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
}

void manageRunway(struct Queue* q, int timeSlots) {
    for (int i = 0; i < timeSlots; i++) {
        printf("\nTime Slot %d:\n", i + 1);
        if (!isQueueEmpty(q)) {
            dequeue(q);
        } else {
            printf("Runway is idle.\n");
        }
    }
}

//9.

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

struct StockOrder {
    char type[4];  // "buy" or "sell"
    int quantity;
    float price;
    int orderID;
};

struct Queue {
    int size;
    int front;
    int rear;
    struct StockOrder *orders;
};

void create(struct Queue *, int);
void enqueue(struct Queue *, struct StockOrder);
void display(struct Queue);
struct StockOrder dequeue(struct Queue *);
int cancelOrder(struct Queue *, int);

int main() {
    struct Queue q;
    create(&q, 5);

    struct StockOrder order1 = {"buy", 100, 50.75, 1};
    struct StockOrder order2 = {"sell", 50, 51.50, 2};
    struct StockOrder order3 = {"buy", 200, 49.25, 3};
    struct StockOrder order4 = {"sell", 150, 52.00, 4};

    enqueue(&q, order1);
    enqueue(&q, order2);
    enqueue(&q, order3);
    enqueue(&q, order4);

    display(q);

    int canceledOrderID = 3;
    printf("\nCanceling Order ID: %d\n", canceledOrderID);
    cancelOrder(&q, canceledOrderID);

    display(q);

    struct StockOrder dequeuedOrder = dequeue(&q);
    printf("\nDequeued Order: Type: %s, Quantity: %d, Price: %.2f, Order ID: %d\n",
dequeuedOrder.type, dequeuedOrder.quantity, dequeuedOrder.price, dequeuedOrder.orderID);

    display(q);

    return 0;
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->orders = (struct StockOrder *)malloc(q->size * sizeof(struct StockOrder));
}

int isQueueFull(struct Queue *q) {
    return q->rear == q->size - 1;
}

int isQueueEmpty(struct Queue *q) {
    return q->front == q->rear;
}

void enqueue(struct Queue *q, struct StockOrder newOrder) {
    if (isQueueFull(q)) {
        printf("Queue is full, cannot enqueue order.\n");
    } else {
        q->rear++;
        q->orders[q->rear] = newOrder;
        printf("Order enqueued: Type: %s, Quantity: %d, Price: %.2f, Order ID: %d\n", newOrder.type,
newOrder.quantity, newOrder.price, newOrder.orderID);
    }
}

struct StockOrder dequeue(struct Queue *q) {
    struct StockOrder emptyOrder = {"", 0, 0.0, 0};
    if (isQueueEmpty(q)) {
        printf("Queue is empty, cannot dequeue order.\n");
    } else {
        q->front++;
        return q->orders[q->front];
    }
    return emptyOrder;
}

void display(struct Queue q) {
    if (isQueueEmpty(&q)) {
        printf("Queue is empty.\n");
    } else {
        printf("Orders in the queue (from front to rear):\n");
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Type: %s | Quantity: %d | Price: %.2f | Order ID: %d\n", q.orders[i].type,
q.orders[i].quantity, q.orders[i].price, q.orders[i].orderID);
        }
        printf("\n");
    }
}

int cancelOrder(struct Queue *q, int orderID) {
    if (isQueueEmpty(q)) {
        printf("Queue is empty, cannot cancel order.\n");
        return 0;
    }

    for (int i = q->front + 1; i <= q->rear; i++) {
        if (q->orders[i].orderID == orderID) {
            for (int j = i; j < q->rear; j++) {
                q->orders[j] = q->orders[j + 1];
            }
            q->rear--;
            printf("Order with ID %d has been canceled.\n", orderID);
            return 1;
        }
    }

    printf("Order with ID %d not found.\n", orderID);
    return 0;
}

//10.

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

struct Attendee {
    int id;
    char name[50];
    struct Attendee* next;
};

struct Queue {
    struct Attendee* front;
    struct Attendee* rear;
};

void initializeQueue(struct Queue* q);
void enqueue(struct Queue* q, int id, const char* name);
void dequeue(struct Queue* q);
void cancelRegistration(struct Queue* q, int id);
void displayQueue(struct Queue* q);

int main() {
    struct Queue conferenceQueue;
    initializeQueue(&conferenceQueue);

    enqueue(&conferenceQueue, 1, "Alice");
    enqueue(&conferenceQueue, 2, "Bob");
    enqueue(&conferenceQueue, 3, "Charlie");
    displayQueue(&conferenceQueue);

    cancelRegistration(&conferenceQueue, 2);
    displayQueue(&conferenceQueue);

    return 0;
}

void initializeQueue(struct Queue* q) {
    q->front = NULL;
    q->rear = NULL;
}

void enqueue(struct Queue* q, int id, const char* name) {
    struct Attendee* newAttendee = (struct Attendee*)malloc(sizeof(struct Attendee));
    if (!newAttendee) {
        printf("Memory allocation failed\n");
        return;
    }

    newAttendee->id = id;
    strncpy(newAttendee->name, name, sizeof(newAttendee->name) - 1);
    newAttendee->name[sizeof(newAttendee->name) - 1] = '\0';
    newAttendee->next = NULL;

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

    printf("Attendee %s (ID: %d) added to the conference queue.\n", name, id);
}

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

    struct Attendee* temp = q->front;
    printf("Serving attendee %s (ID: %d)\n", temp->name, temp->id);
    q->front = q->front->next;

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

    free(temp);
}

void cancelRegistration(struct Queue* q, int id) {
    struct Attendee* temp = q->front;
    struct Attendee* prev = NULL;

    while (temp != NULL && temp->id != id) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Attendee with ID %d not found.\n", id);
        return;
    }

    if (prev == NULL) {
        q->front = temp->next;
    } else {
        prev->next = temp->next;
    }

    if (temp == q->rear) {
        q->rear = prev;
    }

    free(temp);
    printf("Registration for Attendee ID %d canceled.\n", id);
}

void displayQueue(struct Queue* q) {
    struct Attendee* temp = q->front;

    if (temp == NULL) {
        printf("No attendees in the queue.\n");
        return;
    }
    printf("Current Conference Registrations:\n");
    while (temp != NULL) {
        printf("ID: %d, Name: %s\n", temp->id, temp->name);
        temp = temp->next;
    }
}

//11.
#include <stdio.h>
#include <stdlib.h>

#define MEDIA 1
#define GENERAL 0

struct Audience {
    int id;
    int priority;
};

struct Queue {
    int size;
    int front;
    int rear;
    struct Audience *audience;
};

void createQueue(struct Queue *, int);
void enqueue(struct Queue *, struct Audience);
void displayQueue(struct Queue);
struct Audience dequeue(struct Queue *);
```

```
int main() {
    struct Queue q;
    createQueue(&q, 5);

    struct Audience a1 = {101, GENERAL};
    struct Audience a2 = {102, MEDIA};
    struct Audience a3 = {103, GENERAL};

    enqueue(&q, a1);
    enqueue(&q, a2);
    enqueue(&q, a3);

    printf("Audience List:\n");
    displayQueue(q);

    printf("\nDequeue: %d\n", dequeue(&q).id);
    displayQueue(q);

    return 0;
}

void createQueue(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->audience = (struct Audience *)malloc(q->size * sizeof(struct Audience));
}

void enqueue(struct Queue *q, struct Audience a) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->audience[q->rear] = a;
    }
}

struct Audience dequeue(struct Queue *q) {
    struct Audience a = {0, 0};
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        a = q->audience[q->front];
    }
    return a;
}

void displayQueue(struct Queue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
        return;
    }
    for (int i = q.front + 1; i <= q.rear; i++) {
        struct Audience a = q.audience[i];
        printf("ID: %d, Priority: %s\n", a.id, a.priority == MEDIA ? "Media" : "General");
    }
}


//12.

#include <stdio.h>
```

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

struct LoanApplication {
    int id;
    char name[50];
    float loanAmount;
    int creditScore;
    struct LoanApplication* next;
};

struct Queue {
    struct LoanApplication* front;
    struct LoanApplication* rear;
};

void initializeQueue(struct Queue* q);
void enqueue(struct Queue* q, int id, const char* name, float loanAmount, int creditScore);
void dequeue(struct Queue* q);
void cancelApplication(struct Queue* q, int id);
void displayQueue(struct Queue* q);

int main() {
    struct Queue loanQueue;
    initializeQueue(&loanQueue);

    enqueue(&loanQueue, 1, "Sofi", 50000.0, 720);
    enqueue(&loanQueue, 2, "Mickey", 100000.0, 680);
    enqueue(&loanQueue, 3, "vega", 30000.0, 750);
    enqueue(&loanQueue, 4, "Swetha", 40000.0, 690);
    printf("\nInitial Loan Application Queue:\n");
    displayQueue(&loanQueue);

    cancelApplication(&loanQueue, 2);

    printf("\nAfter Canceling Application for Bob:\n");
    displayQueue(&loanQueue);

    printf("\nProcessing Loan Applications:\n");
    while (loanQueue.front != NULL) {
        dequeue(&loanQueue);
    }

    return 0;
}

void initializeQueue(struct Queue* q) {
    q->front = NULL;
    q->rear = NULL;
}

void enqueue(struct Queue* q, int id, const char* name, float loanAmount, int creditScore) {
    struct LoanApplication* newApp = (struct LoanApplication*)malloc(sizeof(struct LoanApplication));
    if (!newApp) {
        printf("Memory allocation failed!\n");
        return;
    }

    newApp->id = id;
    strncpy(newApp->name, name, sizeof(newApp->name) - 1);
    newApp->name[sizeof(newApp->name) - 1] = '\0';
    newApp->loanAmount  = loanAmount;

    newApp->creditScore  = creditScore;

    newApp->next  = NULL;


    if (q->front  == NULL ||

        (q->front->loanAmount  < loanAmount)  ||

        (q->front->loanAmount  == loanAmount  && q->front->creditScore  < creditScore)) {

        newApp->next  = q->front;

        q->front  = newApp;

        if (q->rear == NULL) {

            q->rear = newApp;

        }

    } else {

        struct LoanApplication*  current = q->front;

        while  (current->next  != NULL &&

              (current->next->loanAmount  > loanAmount  ||

               (current->next->loanAmount  == loanAmount  && current->next->creditScore  >=
creditScore)))  {

            current = current->next;

        }

        newApp->next  = current->next;

        current->next  = newApp;

        if (newApp->next  == NULL) {

            q->rear = newApp;

        }

    }


    printf("Loan  Application  from  %s (ID: %d) added to the queue.\n", name, id);

}


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

    struct LoanApplication* temp = q->front;
    printf("Processing Loan Application for %s (ID: %d), Loan Amount: %.2f, Credit Score: %d\n",
           temp->name, temp->id, temp->loanAmount, temp->creditScore);

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

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

    free(temp);
}

void cancelApplication(struct Queue* q, int id) {
    struct LoanApplication* temp = q->front;
    struct LoanApplication* prev = NULL;

    while (temp != NULL && temp->id != id) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Loan application with ID %d not found.\n", id);
        return;
    }
    if (prev == NULL) {
        q->front = temp->next;
    } else {
        prev->next = temp->next;
    }

    if (temp == q->rear) {
        q->rear = prev;
    }

    free(temp);
    printf("Loan Application for %d canceled.\n", id);
}

void displayQueue(struct Queue* q) {
    struct LoanApplication* temp = q->front;

    if (temp == NULL) {
        printf("No loan applications in the queue.\n");
        return;
    }

    printf("Loan Applications in the Queue:\n");
    while (temp != NULL) {
        printf("ID: %d, Name: %s, Loan Amount: %.2f, Credit Score: %d\n",
               temp->id, temp->name, temp->loanAmount, temp->creditScore);
        temp = temp->next;
    }
}
```

```
//13.

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

struct Customer {
    int id;
    int items;
};

struct Queue {
    int size;
    int front;
    int rear;
    struct Customer *customers;
};

void createQueue(struct Queue *, int);
void enqueue(struct Queue *, struct Customer);
void displayQueue(struct Queue);
struct Customer dequeue(struct Queue *);

int main() {
    struct Queue q;
    createQueue(&q, 5);

    struct Customer c1 = {101, 2};
    struct Customer c2 = {102, 5};
    struct Customer c3 = {103, 3};

    enqueue(&q, c1);
    enqueue(&q, c2);
    enqueue(&q, c3);

    printf("Checkout Queue:\n");
    displayQueue(q);

    printf("\nDequeue: %d\n", dequeue(&q).id);
    displayQueue(q);

    return 0;
}

void createQueue(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->customers = (struct Customer *)malloc(q->size * sizeof(struct Customer));
}

void enqueue(struct Queue *q, struct Customer c) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->customers[q->rear] = c;
    }
}

struct Customer dequeue(struct Queue *q) {
    struct Customer c = {0, 0};
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        c = q->customers[q->front];
    }
    return c;
}

void displayQueue(struct Queue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
        return;
    }
    for (int i = q.front + 1; i <= q.rear; i++) {
        struct Customer c = q.customers[i];
        printf("Customer ID: %d, Items: %d\n", c.id, c.items);
    }
}



//14.

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

struct Bus {
    int id;
    char type[20];
    int arrivalTime;
    int priority; // Priority: 1 for express buses, 0 for normal buses
    struct Bus* next;
```

```
};

struct Queue {
    struct Bus* front;
    struct Bus* rear;
};

void initializeQueue(struct Queue* q);
void enqueue(struct Queue* q, int id, const char* type, int arrivalTime, int priority);
void dequeue(struct Queue* q);
void displayQueue(struct Queue* q);
void processBuses(struct Queue* q, int peakHours);

int main() {
    struct Queue busQueue;
    initializeQueue(&busQueue);

    enqueue(&busQueue, 1, "Normal", 900, 0);  // 9:00 AM
    enqueue(&busQueue, 2, "Express", 1000, 1);  // 10:00 AM
    enqueue(&busQueue, 3, "Normal", 1100, 0);  // 11:00 AM
    enqueue(&busQueue, 4, "Express", 1200, 1);  // 12:00 PM

    printf("\nInitial Bus Queue:\n");
    displayQueue(&busQueue);

    printf("\nProcessing buses during peak hours (8 AM to 10 AM):\n");
    processBuses(&busQueue, 1);

    printf("\nProcessing buses during off-peak hours (after 10 AM):\n");
    processBuses(&busQueue, 0);
    return 0;
}

void initializeQueue(struct Queue* q) {
    q->front = NULL;
    q->rear = NULL;
}

void enqueue(struct Queue* q, int id, const char* type, int arrivalTime, int priority) {
    struct Bus* newBus = (struct Bus*)malloc(sizeof(struct Bus));
    if (!newBus) {
        printf("Memory allocation failed!\n");
        return;
    }

    newBus->id = id;
    strncpy(newBus->type, type, sizeof(newBus->type) - 1);
    newBus->type[sizeof(newBus->type) - 1] = '\0';
    newBus->arrivalTime = arrivalTime;
    newBus->priority = priority;
    newBus->next = NULL;

    if (q->front == NULL || q->front->priority < priority) {
        newBus->next = q->front;
        q->front = newBus;
        if (q->rear == NULL) {
            q->rear = newBus;
        }
    } else {
        struct Bus* current = q->front;
        while (current->next != NULL && current->next->priority >= priority) {
            current = current->next;
        }
        newBus->next = current->next;
        current->next = newBus;
        if (newBus->next == NULL) {
            q->rear = newBus;
        }
    }
    printf("Bus %d (%s) arrived at %d\n", id, type, arrivalTime);
}

void dequeue(struct Queue* q) {
    if (q->front == NULL) {
        printf("No buses in the queue.\n");
        return;
    }

    struct Bus* temp = q->front;
    printf("Bus %d (%s) departing at %d\n", temp->id, temp->type, temp->arrivalTime);
    q->front = q->front->next;

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

    free(temp);
}

void displayQueue(struct Queue* q) {
    struct Bus* temp = q->front;
    if (temp == NULL) {
        printf("No buses in the queue.\n");
        return;
    }

    printf("Current Bus Queue:\n");
    while (temp != NULL) {
        printf("ID: %d, Type: %s, Arrival Time: %d, Priority: %d\n",
               temp->id, temp->type, temp->arrivalTime, temp->priority);
        temp = temp->next;
    }
}

void processBuses(struct Queue* q, int peakHours) {
    struct Bus* temp = q->front;
    if (peakHours == 1) {
        printf("Processing buses during peak hours.\n");
    } else {
        printf("Processing buses during off-peak hours.\n");
    }

    while (temp != NULL) {
        if ((peakHours == 1 && temp->arrivalTime >= 800 && temp->arrivalTime <= 1000) ||
            (peakHours == 0 && temp->arrivalTime > 1000)) {
            dequeue(q);
        }
        temp = temp->next;
    }
}

//15.
```

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

#define VIP 1
#define REGULAR 0

struct Attendee {
   int id;
   int type; // VIP or Regular
};

struct Queue {
   int size;
   int front;
   int rear;
   struct Attendee *attendees;
};

void createQueue(struct  Queue *, int);
void enqueue(struct  Queue *, struct Attendee);
void displayQueue(struct  Queue);
struct Attendee dequeue(struct  Queue *);

int main() {
   struct Queue q;
   createQueue(&q,  5);

   struct Attendee a1 = {101, REGULAR};
   struct Attendee a2 = {102, VIP};
   struct Attendee a3 = {103, REGULAR};
```

```
    enqueue(&q, a1);
    enqueue(&q, a2);
    enqueue(&q, a3);

    printf("Rally Attendees:\n");
    displayQueue(q);

    printf("\nDequeue: %d\n", dequeue(&q).id);
    displayQueue(q);

    return 0;
}

void createQueue(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->attendees = (struct Attendee *)malloc(q->size * sizeof(struct Attendee));
}

void enqueue(struct Queue *q, struct Attendee a) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->attendees[q->rear] = a;
    }
}

struct Attendee dequeue(struct Queue *q) {
    struct Attendee a = {0, 0};
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        a = q->attendees[q->front];
    }
    return a;
}

void displayQueue(struct Queue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
        return;
    }
    for (int i = q.front + 1; i <= q.rear; i++) {
        struct Attendee a = q.attendees[i];
        printf("ID: %d, Type: %s\n", a.id, a.type == VIP ? "VIP" : "Regular");
    }
}


//16.

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

struct Transaction {
    int transactionId;
    char type[20]; // Deposit, Withdrawal, Transfer
    double amount;
    int fromAccountId;
    int toAccountId;
    struct Transaction* next;
};

struct Queue {
    struct Transaction* front;
    struct Transaction* rear;
};

void initializeQueue(struct Queue* q);
void enqueue(struct Queue* q, int transactionId, const char* type, double amount, int
fromAccountId, int toAccountId);
void dequeue(struct Queue* q);
void displayQueue(struct Queue* q);
void processTransaction(struct Queue* q);

int main() {
    struct Queue transactionQueue;
    initializeQueue(&transactionQueue);

    enqueue(&transactionQueue, 1, "Deposit", 1000.0, 12345, -1);
    enqueue(&transactionQueue, 2, "Withdrawal", 500.0, 12345, -1);
    enqueue(&transactionQueue, 3, "Transfer", 200.0, 12345, 54321);

    printf("Initial Transaction Queue:\n");
    displayQueue(&transactionQueue);

    printf("\nProcessing Transactions:\n");
    processTransaction(&transactionQueue);

    return 0;
```

```
}

void initializeQueue(struct Queue* q) {
    q->front = NULL;
    q->rear = NULL;
}

void enqueue(struct Queue* q, int transactionId, const char* type, double amount, int
fromAccountId, int toAccountId) {
    struct Transaction* newTransaction = (struct Transaction*)malloc(sizeof(struct Transaction));
    if (!newTransaction) {
        printf("Memory allocation failed!\n");
        return;
    }

    newTransaction->transactionId = transactionId;
    strncpy(newTransaction->type, type, sizeof(newTransaction->type) - 1);
    newTransaction->type[sizeof(newTransaction->type) - 1] = '\0';
    newTransaction->amount = amount;
    newTransaction->fromAccountId = fromAccountId;
    newTransaction->toAccountId = toAccountId;
    newTransaction->next = NULL;

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

    printf("Transaction %d (%s) added to the queue: Amount = %.2f, From Account = %d, To Account =
%d\n",
        transactionId, type, amount, fromAccountId, toAccountId);
}

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

    struct Transaction* temp = q->front;
    printf("Processing Transaction %d (%s): Amount = %.2f, From Account = %d, To Account = %d\n",
        temp->transactionId, temp->type, temp->amount, temp->fromAccountId, temp-
>toAccountId);

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

    free(temp);
}

void displayQueue(struct Queue* q) {
    struct Transaction* temp = q->front;

    if (temp == NULL) {
        printf("No transactions in the queue.\n");
        return;
    }

    printf("Transaction Queue:\n");
    while (temp != NULL) {
        printf("Transaction ID: %d, Type: %s, Amount: %.2f, From Account: %d, To Account: %d\n",
                temp->transactionId, temp->type, temp->amount, temp->fromAccountId, temp-
>toAccountId);
        temp = temp->next;
    }
}

void processTransaction(struct Queue* q) {
    while (q->front != NULL) {
        dequeue(q);
    }
}

//17.

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

struct Voter {
    int id;
    int verified; // 1 for verified, 0 for unverified
};

struct Queue {
    int size;
    int front;
    int rear;
    struct Voter *voters;
};
```

```
void createQueue(struct Queue *, int);
void enqueue(struct Queue *, struct Voter);
void displayQueue(struct Queue);
struct Voter dequeue(struct Queue *);

int main() {
    struct Queue q;
    createQueue(&q, 5);

    struct Voter v1 = {101, 1};
    struct Voter v2 = {102, 0};
    struct Voter v3 = {103, 1};

    enqueue(&q, v1);
    enqueue(&q, v2);
    enqueue(&q, v3);

    printf("Voters Queue:\n");
    displayQueue(q);

    printf("\nDequeue: %d\n", dequeue(&q).id);
    displayQueue(q);

    return 0;
}

void createQueue(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->voters = (struct Voter *)malloc(q->size * sizeof(struct Voter));
}
```

```
void enqueue(struct  Queue *q, struct Voter v) {
    if (q->rear == q->size - 1) {
        printf("Queue  is full\n");
    } else {
        q->rear++;
        q->voters[q->rear] = v;
    }
}

struct Voter dequeue(struct  Queue *q) {
    struct Voter v = {0, 0};
    if (q->front == q->rear) {
        printf("Queue  is empty\n");
    } else {
        q->front++;
        v = q->voters[q->front];
    }
    return v;
}

void displayQueue(struct  Queue q) {
    if (q.front == q.rear) {
        printf("Queue  is empty\n");
        return;
    }
    for (int i = q.front + 1; i <= q.rear; i++) {
        struct Voter v = q.voters[i];
        printf("ID: %d, Verified: %s\n", v.id, v.verified == 1 ? "Yes" : "No");
    }
}
```

```
//18.

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

struct Patient {
    int id;
    char name[50];
    int severity;
    struct Patient* next;
};

struct Queue {
    struct Patient* front;
    struct Patient* rear;
};

void initializeQueue(struct Queue* q);
void enqueue(struct Queue* q, int id, const char* name, int severity);
void dequeue(struct Queue* q);
void displayQueue(struct Queue* q);
void processPatient(struct Queue* q, int doctorId);

int main() {
    struct Queue emergencyQueue;
    initializeQueue(&emergencyQueue);

    enqueue(&emergencyQueue, 1, "Alice", 5);
    enqueue(&emergencyQueue, 2, "Bob", 9);
    enqueue(&emergencyQueue, 3, "Charlie", 3);
    enqueue(&emergencyQueue, 4, "David", 8);

    printf("Initial Patient Queue:\n");
    displayQueue(&emergencyQueue);

    printf("\nProcessing Patients with 2 Doctors:\n");
    processPatient(&emergencyQueue, 1);
    processPatient(&emergencyQueue, 2);

    return 0;
}

void initializeQueue(struct Queue* q) {
    q->front = NULL;
    q->rear = NULL;
}

void enqueue(struct Queue* q, int id, const char* name, int severity) {
    struct Patient* newPatient = (struct Patient*)malloc(sizeof(struct Patient));
    if (!newPatient) {
        printf("Memory allocation failed!\n");
        return;
    }

    newPatient->id = id;
    strncpy(newPatient->name, name, sizeof(newPatient->name) - 1);
    newPatient->name[sizeof(newPatient->name) - 1] = '\0';
    newPatient->severity = severity;
    newPatient->next = NULL;
    if (q->front == NULL || severity >= q->front->severity) {
        newPatient->next = q->front;
        q->front = newPatient;
        if (q->rear == NULL) {
            q->rear = newPatient;
        }
    } else {
        struct Patient* temp = q->front;
        while (temp->next != NULL && temp->next->severity >= severity) {
            temp = temp->next;
        }
        newPatient->next = temp->next;
        temp->next = newPatient;
        if (newPatient->next == NULL) {
            q->rear = newPatient;
        }
    }

    printf("Patient %s (ID: %d, Severity: %d) added to the queue.\n", name, id, severity);
}

void dequeue(struct Queue* q) {
    if (q->front == NULL) {
        printf("No patients in the queue.\n");
        return;
    }

    struct Patient* temp = q->front;
    printf("Patient %s (ID: %d, Severity: %d) is being treated.\n", temp->name, temp->id, temp-
>severity);
```

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

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

    free(temp);
}

void displayQueue(struct Queue* q) {
    struct Patient* temp = q->front;
    if (temp == NULL) {
        printf("No patients in the queue.\n");
        return;
    }

    printf("Current Patient Queue (by Severity):\n");
    while (temp != NULL) {
        printf("ID: %d, Name: %s, Severity: %d\n", temp->id, temp->name, temp->severity);
        temp = temp->next;
    }
}

void processPatient(struct Queue* q, int doctorId) {
    printf("\nDoctor %d is attending to the patient.\n", doctorId);
    dequeue(q);
}

//19.

#include <stdio.h>
```

```
#include <stdlib.h>

struct Surveyor {
   int id;
   int surveysCollected;
};

struct Queue {
   int size;
   int front;
   int rear;
   struct Surveyor *surveyors;
};

void createQueue(struct Queue *, int);
void enqueue(struct Queue *, struct Surveyor);
void displayQueue(struct Queue);
struct Surveyor dequeue(struct Queue *);

int main() {
   struct Queue q;
   createQueue(&q, 5);

   struct Surveyor s1 = {101, 5};
   struct Surveyor s2 = {102, 3};
   struct Surveyor s3 = {103, 8};

   enqueue(&q, s1);
   enqueue(&q, s2);
   enqueue(&q, s3);
```

```
    printf("Surveyors Queue:\n");
    displayQueue(q);

    printf("\nDequeue: %d\n", dequeue(&q).id);
    displayQueue(q);

    return 0;
}

void createQueue(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->surveyors = (struct Surveyor *)malloc(q->size * sizeof(struct Surveyor));
}

void enqueue(struct Queue *q, struct Surveyor s) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->surveyors[q->rear] = s;
    }
}

struct Surveyor dequeue(struct Queue *q) {
    struct Surveyor s = {0, 0};
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        s = q->surveyors[q->front];
    }
    return s;
}

void displayQueue(struct Queue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
        return;
    }
    for (int i = q.front + 1; i <= q.rear; i++) {
        struct Surveyor s = q.surveyors[i];
        printf("ID: %d, Surveys Collected: %d\n", s.id, s.surveysCollected);
    }
}

//20.

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

struct StockData {
    int id;
    char symbol[10];
    float price;
    struct StockData* next;
};

struct Queue {
    struct StockData* front;
    struct StockData* rear;
    int size;
};

void initializeQueue(struct Queue* q);
int isQueueEmpty(struct Queue* q);
void enqueue(struct Queue* q, int id, const char* symbol, float price);
void dequeue(struct Queue* q);
void displayQueue(struct Queue* q);
void analyzeData(struct Queue* q);
float calculateAveragePrice(struct Queue* q);

int main() {
    struct Queue marketDataQueue;
    initializeQueue(&marketDataQueue);

    enqueue(&marketDataQueue, 1, "AAPL", 150.25);
    enqueue(&marketDataQueue, 2, "GOOG", 2750.50);
    enqueue(&marketDataQueue, 3, "AMZN", 3400.75);
    enqueue(&marketDataQueue, 4, "TSLA", 720.10);

    printf("Market Data Queue:\n");
    displayQueue(&marketDataQueue);

    analyzeData(&marketDataQueue);

    dequeue(&marketDataQueue);
    dequeue(&marketDataQueue);

    printf("\nMarket Data Queue After Dequeue:\n");
    displayQueue(&marketDataQueue);
    analyzeData(&marketDataQueue);


    return 0;
}


void initializeQueue(struct Queue* q) {
    q->front = NULL;
    q->rear = NULL;
    q->size = 0;
}


int isQueueEmpty(struct Queue* q) {
    return q->front == NULL;
}


void enqueue(struct Queue* q, int id, const char* symbol, float price) {
    struct StockData* newData = (struct StockData*)malloc(sizeof(struct StockData));
    if (!newData) {
        printf("Memory allocation failed!\n");
        return;
    }


    newData->id = id;
    strncpy(newData->symbol, symbol, sizeof(newData->symbol) - 1);
    newData->symbol[sizeof(newData->symbol) - 1] = '\0';
    newData->price = price;
    newData->next = NULL;


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

    q->size++;
    printf("Market Data (%s - ID: %d) with price %.2f added to the queue.\n", symbol, id, price);
}

void dequeue(struct Queue* q) {
    if (isQueueEmpty(q)) {
        printf("No market data to process.\n");
        return;
    }

    struct StockData* temp = q->front;
    printf("Processing Market Data (%s - ID: %d) with price %.2f.\n", temp->symbol, temp->id, temp-
>price);
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    q->size--;
}

void displayQueue(struct Queue* q) {
    if (isQueueEmpty(q)) {
        printf("No market data in the queue.\n");
        return;
    }
    struct StockData* temp = q->front;
    while (temp != NULL) {
        printf("ID: %d, Symbol: %s, Price: %.2f\n", temp->id, temp->symbol, temp->price);
        temp = temp->next;
    }
}

void analyzeData(struct Queue* q) {
    if (isQueueEmpty(q)) {
        printf("No data to analyze.\n");
        return;
    }

    printf("\nAnalyzing Market Data...\n");

    float averagePrice = calculateAveragePrice(q);
    printf("Average Price of the current data: %.2f\n", averagePrice);

    // Example decision-making based on the average price
    if (averagePrice > 1000.00) {
        printf("Market is performing well.\n");
    } else {
        printf("Market is underperforming.\n");
    }
}

float calculateAveragePrice(struct Queue* q) {
    float total = 0.0;
    int count = 0;
    struct StockData* temp = q->front;
    while (temp != NULL) {
        total += temp->price;
        count++;
        temp = temp->next;
    }

    if (count == 0) return 0.0;
    return total / count;
}
```