---Queue using linkedlist---

1.

stock market

```c
#include <stdio.h>

#include <stdlib.h>


struct order {

    char type; // 'B' for Buy, 'S' for Sell

    float price;

    int quantity;

};


struct queue {

    int size;

    int front;

    int rear;

    struct order *q;

};


void createQueue(struct queue *qu, int size) {

    qu->size = size;

    qu->front = -1;

    qu->rear = -1;

    qu->q = (struct order *)malloc(size * sizeof(struct order));

}


int isFull(struct queue *qu) {

    return qu->rear == qu->size - 1;

}
```

```
int isEmpty(struct queue *qu) {
    return qu->front == qu->rear;
}

void enqueue(struct queue *qu, struct order data) {
    if (isFull(qu)) {
        printf("Queue is full! Cannot add more orders.\n");
    } else {
        qu->rear++;
        qu->q[qu->rear] = data;
    }
}

struct order dequeue(struct queue *qu) {
    if (isEmpty(qu)) {
        printf("Queue is empty! No orders to process.\n");
        struct order emptyOrder = {'\0', 0, 0};
        return emptyOrder;
    } else {
        qu->front++;
        return qu->q[qu->front];
    }
}

void display(struct queue qu) {
    if (isEmpty(&qu)) {
        printf("Queue is empty.\n");
    } else {
        for (int i = qu.front + 1; i <= qu.rear; i++) {
            printf("%c Order - Price: %.2f, Quantity: %d\n",
                   qu.q[i].type, qu.q[i].price, qu.q[i].quantity);
        }
    }
}

void matchOrders(struct queue *buyQueue, struct queue *sellQueue) {
    while (!isEmpty(buyQueue) && !isEmpty(sellQueue)) {
        struct order buyOrder = buyQueue->q[buyQueue->front + 1];
        struct order sellOrder = sellQueue->q[sellQueue->front + 1];

        if (buyOrder.price >= sellOrder.price) {
            int matchedQuantity = buyOrder.quantity < sellOrder.quantity ? buyOrder.quantity :
sellOrder.quantity;
            printf("Matched: Buy %.2f x %d with Sell %.2f x %d\n",
                   buyOrder.price, matchedQuantity, sellOrder.price, matchedQuantity);

            buyQueue->q[buyQueue->front + 1].quantity -= matchedQuantity;
            sellQueue->q[sellQueue->front + 1].quantity -= matchedQuantity;

            if (buyQueue->q[buyQueue->front + 1].quantity == 0) {
                dequeue(buyQueue);
            }
            if (sellQueue->q[sellQueue->front + 1].quantity == 0) {
                dequeue(sellQueue);
            }
        } else {
            break;
        }
    }
}

int main() {
    struct queue buyQueue, sellQueue;
    int size = 5;

    createQueue(&buyQueue, size);
    createQueue(&sellQueue, size);

    struct order order1 = {'B', 100.0, 10};
    struct order order2 = {'S', 98.0, 5};
    struct order order3 = {'B', 102.0, 15};
    struct order order4 = {'S', 99.0, 10};

    enqueue(&buyQueue, order1);
    enqueue(&sellQueue, order2);
    enqueue(&buyQueue, order3);
    enqueue(&sellQueue, order4);

    printf("Buy Orders:\n");
    display(buyQueue);
    printf("\nSell Orders:\n");
    display(sellQueue);

    printf("\nMatching Orders:\n");
    matchOrders(&buyQueue, &sellQueue);
    printf("\nRemaining Buy Orders:\n");
    display(buyQueue);
    printf("\nRemaining Sell Orders:\n");
    display(sellQueue);

    free(buyQueue.q);
    free(sellQueue.q);

    return 0;
}
```

2.political campaign event management

```
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#define MAX 10

struct attendee {
    char name[50];
    int isVIP;
};

struct queue {
    int front, rear;
    struct attendee data[MAX];
};

void initQueue(struct queue *q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(struct queue *q) {
    return q->rear == MAX - 1;
}

int isEmpty(struct queue *q) {
    return q->front == q->rear;
}

void enqueue(struct queue *q, struct attendee person) {
    if (isFull(q)) {
        printf("Queue is full! Cannot add more attendees.\n");
    } else {
        q->rear++;
        q->data[q->rear] = person;
    }
}
void priorityEnqueue(struct queue *q, struct attendee person) {
    if (isFull(q)) {
        printf("Queue is full! Cannot add more attendees.\n");
    } else {
        for (int i = q->rear; i >= q->front + 1; i--) {
            q->data[i + 1] = q->data[i];
        }
        q->data[q->front + 1] = person;
        q->rear++;
    }
}

struct attendee dequeue(struct queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty! No attendees to check in.\n");
        struct attendee empty = {"", 0};
        return empty;
    } else {
        q->front++;
        return q->data[q->front];
    }
}

void display(struct queue q) {
    if (isEmpty(&q)) {
        printf("Queue is empty.\n");
    } else {
        printf("Attendee List:\n");
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("%s (VIP: %s)\n", q.data[i].name, q.data[i].isVIP ? "Yes" : "No");
        }
    }
}

int main() {
    struct queue eventQueue;
    initQueue(&eventQueue);
    struct attendee att1 = {"Alice", 0};
    struct attendee att2 = {"Bob", 1};
    struct attendee att3 = {"Charlie", 0};
    struct attendee att4 = {"Diana", 1};
    enqueue(&eventQueue, att1);
    priorityEnqueue(&eventQueue, att2);
    enqueue(&eventQueue, att3);
    priorityEnqueue(&eventQueue, att4);
    printf("Queue after registrations:\n");
    display(eventQueue);
    printf("\nChecking in attendees:\n");
    while (!isEmpty(&eventQueue)) {
        struct attendee checkedIn = dequeue(&eventQueue);
        printf("Checked in: %s (VIP: %s)\n", checkedIn.name, checkedIn.isVIP ? "Yes" : "No");
    }

    return 0;
}
```

3.political debut

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct details{
    int entry;
    int is_media;
    char seating[30];
};
struct queue{
    int size;
    int front;

    int rear;

    struct details *q;

};

void initQueue(struct queue *qu,int size){

    qu->size=size;

    qu->front=qu->rear=-1;

    qu->q=(struct details*)malloc(sizeof(struct details));

}

void enqueue(struct queue *qu,struct details dt1){

    if(qu->rear==qu->size-1){

        printf("the queue is full!");

    }else{

        qu->rear+=1;

        qu->q[qu->rear]=dt1;

    }

}

void priorityqueue(struct queue *qu,struct details dt1){

    if (qu->rear == qu->size - 1) {

        printf("The queue is full!\n");

    }else{

        for(int i=qu->rear;i>=qu->front;i--){

            qu->q[i+1]=qu->q[i];

        }

        qu->q[qu->front+1]=dt1;

        qu->rear+=1;

    }

}

struct details dequeue(struct queue *qu){
    if(qu->front==qu->rear){
        printf("The queue is empty");
    }else{
        qu->front+=1;
        return qu->q[qu->front];
    }
}
void display(struct queue qu){
    printf("list of memebers\n");
    for(int i=qu.front+1;i<=qu.rear;i++){
        printf("Entry permitted : %s Media:%s
Seating:%s",qu.q[i].entry?"Yes":"No",qu.q[i].is_media?"Yes":"No",qu.q[i].seating);
    }
}
int main(){
    struct queue qu;
    initQueue(&qu,5);
    struct details dt1={1,0,"back"};
    struct details dt2={1,1,"front"};
    struct details dt3={0,0,"Not permitted"};
    enqueue(&qu,dt1);
    priorityqueue(&qu,dt2);
    enqueue(&qu,dt3);
    display(qu);
    printf("processing attendees");
    while(qu.front!=qu.rear){
        struct details processed=dequeue(&qu);
        printf("Processed: Entry permitted: %s, Media: %s, Seating: %s\n",
               processed.entry ? "Yes" : "No",
            processed.is_media ? "Yes" : "No",
            processed.seating);
   }
   return 0;
   }
}
```

4. customer center simulation

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct node{
   char name[50];
   int priority;
   struct node *next;
}*front=NULL,*rear=NULL;

void enqueue(char *name,int priority){
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   if(temp==NULL){
      printf("Queue is full");
   }else{
      strcpy(temp->name,name);
      temp->priority=priority;
      temp->next=NULL;
      if(front==NULL){
         front=rear=temp;
      }else if(priority==1){
            temp->next=front;
            front=temp;
        }else{
            rear->next=temp;
            rear=temp;
        }
    }
}
void dequeue(){
    struct node *temp=front;
    while(front==NULL){
        printf("Nothing to display ");
    }
    front=front->next;
    if(front==NULL){
        rear=NULL;
    }
    free(temp);
}
void display(){
    struct node *temp=front;
    while(temp!=NULL){
        printf("Name :%s Priority :%s\n",temp->name,temp->priority?"VIP":"Regular");
        temp=temp->next;
    }
    free(temp);
}

int main(){
    enqueue("akshay",1);
    enqueue("abhi",0);
    enqueue("someone",0);
    display();
    dequeue();
    printf("after dequeue\n");
    display();
    return 0;

}
```

5.real time data feed processing

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

#define MAX_SIZE 100

typedef struct {
    int data[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}

int isEmpty(Queue *q) {
    return q->size == 0;
}

void enqueue(Queue *q, int value) {
    if (isFull(q)) {
        printf("Queue is full. Cannot add more data.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->data[q->rear] = value;
    q->size++;
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    int value = q->data[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return value;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = 0; i < q->size; i++) {
        printf("%d ", q->data[(q->front + i) % MAX_SIZE]);
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);

    displayQueue(&q);

    printf("Dequeued: %d\n", dequeue(&q));
    displayQueue(&q);

    return 0;
}
```

6.traffic light control system

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

```c
#include <stdlib.h>

#define MAX_SIZE 100

typedef struct {
    int data[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}

int isEmpty(Queue *q) {
    return q->size == 0;
}

void enqueue(Queue *q, int value) {
    if (isFull(q)) {
        printf("Queue is full. Cannot add more data.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->data[q->rear] = value;
    q->size++;
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    int value = q->data[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return value;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = 0; i < q->size; i++) {
        printf("%d ", q->data[(q->front + i) % MAX_SIZE]);
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);

    displayQueue(&q);

    printf("Dequeued: %d\n", dequeue(&q));
    displayQueue(&q);

    return 0;
}
```

7.election vote counting system

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

#define MAX_SIZE 100

typedef struct {
    int votes[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
```

```
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}

int isEmpty(Queue *q) {
    return q->size == 0;
}

void enqueue(Queue *q, int vote) {
    if (isFull(q)) {
        printf("Queue is full. Cannot accept more votes.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->votes[q->rear] = vote;
    q->size++;
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty. No votes to count.\n");
        return -1;
    }
    int vote = q->votes[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return vote;
```

```
}

void countVotes(Queue *q) {
    if (isEmpty(q)) {
        printf("No votes to count.\n");
        return;
    }
    printf("Counting votes: ");
    while (!isEmpty(q)) {
        printf("%d ", dequeue(q));
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 101);
    enqueue(&q, 102);
    enqueue(&q, 103);

    printf("Votes received.\n");
    countVotes(&q);

    return 0;
}
```

8.airport runway management

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

typedef struct Node {
    int planeId;
    int isEmergency; // 1 for emergency, 0 for normal
    struct Node *next;
} Node;

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

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

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

void enqueue(Queue *q, int planeId, int isEmergency) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->planeId = planeId;
    newNode->isEmergency = isEmergency;
    newNode->next = NULL;

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

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No planes in the queue.\n");
        return -1;
    }
    int planeId = q->front->planeId;
    Node *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return planeId;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No planes waiting.\n");
        return;
    }
    Node *current = q->front;
    printf("Planes in queue: ");
    while (current) {
        printf("%d%s ", current->planeId, current->isEmergency ? "(Emergency)" : "");
        current = current->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 101, 0);
    enqueue(&q, 102,
```

9.stock trading simulation

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

```
#define MAX_SIZE 100

typedef struct {
    int orders[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}

int isEmpty(Queue *q) {
    return q->size == 0;
}

void enqueue(Queue *q, int order) {
    if (isFull(q)) {
        printf("Order queue is full. Cannot place more orders.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->orders[q->rear] = order;
    q->size++;
```

```
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No orders to process.\n");
        return -1;
    }
    int order = q->orders[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return order;
}

void displayOrders(Queue *q) {
    if (isEmpty(q)) {
        printf("No orders in the queue.\n");
        return;
    }
    printf("Pending orders: ");
    for (int i = 0; i < q->size; i++) {
        printf("%d ", q->orders[(q->front + i) % MAX_SIZE]);
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);
    enqueue(&q, 1001); // Buy order
    enqueue(&q, 1002); // Sell order
    enqueue(&q, 1003); // Buy order

    displayOrders(&q);

    printf("Orde
```

10.conference registraion system

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

typedef struct Node {
    char name[50];
    struct Node *next;
} Node;

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

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

int isEmpty(Queue *q) {
    return q->front == NULL;
}
```

```
void enqueue(Queue *q, char *name) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    strcpy(newNode->name, name);
    newNode->next = NULL;

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

char *dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No registrations in the queue.\n");
        return NULL;
    }
    Node *temp = q->front;
    char *name = temp->name;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return name;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No registrations in the queue.\n");
        return;
    }
    Node *current = q->front;
    printf("Registrations: ");
    while (current) {
        printf("%s ", current->name);
        current = current->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, "Alice");
    enqueue(&q, "Bob");
    enqueue(&q, "Charlie");

    displayQueue(&q);

    printf("Processing registration: %s\n", dequeue(&q));
    displayQueue(&q);

    return 0;
}
```

11.political debate audience management

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

#define MAX_SIZE 100

typedef struct {
    int ids[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}

int isEmpty(Queue *q) {
    return q->size == 0;
}
```

```
void enqueue(Queue *q, int id) {
    if (isFull(q)) {
        printf("Audience queue is full. Cannot add more people.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->ids[q->rear] = id;
    q->size++;
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No audience in the queue.\n");
        return -1;
    }
    int id = q->ids[q->front];
    q->front = (
```

12.bank loan applicaton processing

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

typedef struct Node {
    int loanAmount;
    int creditScore;
    struct Node *next;
} Node;
```

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

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

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

void enqueue(Queue *q, int loanAmount, int creditScore) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->loanAmount = loanAmount;
    newNode->creditScore = creditScore;
    newNode->next = NULL;

    if (isEmpty(q)) {
        q->front = q->rear = newNode;
    } else {
        Node *current = q->front, *prev = NULL;
        while (current && current->creditScore >= creditScore) {
            prev = current;
            current = current->next;
        }
        if (prev == NULL) {
            newNode->next = q->front;
            q->front = newNode;
        } else {
            prev->next = newNode;
            newNode->next = current;
        }
        if (current == NULL) {
            q->rear = newNode;
        }
    }
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No loan applications to process.\n");
        return -1;
    }
    Node *temp = q->front;
    int loanAmount = temp->loanAmount;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return loanAmount;
}
```

```
void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No loan applications in the queue.\n");
        return;
    }
    Node *current = q->front;
    printf("Loan Applications: ");
    while (current) {
        printf("[Loan: %d, Score: %d] ", current->loanAmount, current->creditScore);
        current = current->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 50000, 700);
    enqueue(&q, 200000, 800); // High priority
    enqueue(&q, 30000, 650);

    displayQueue(&q);

    printf("Processing loan of amount: %d\n", dequeue(&q));
    displayQueue(&q);

    return 0;
}
```

13.online shopping checkout system

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

#define MAX_SIZE 100

typedef struct {
    int customerId;
    char item[50];
} Order;

typedef struct {
    Order orders[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}

int isEmpty(Queue *q) {
    return q->size == 0;
}

void enqueue(Queue *q, int customerId, char *item) {
    if (isFull(q)) {
        printf("Queue is full. Cannot add more customers.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->orders[q->rear].customerId = customerId;
    strcpy(q->orders[q->rear].item, item);
    q->size++;
}

Order dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No customers in the queue.\n");
        Order emptyOrder = {-1, ""};
        return emptyOrder;
    }
    Order order = q->orders[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return order;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No orders in the queue.\n");
        return;
    }
    printf("Pending orders: ");
    for (int i = 0; i < q->size; i++) {
        printf("[Customer ID: %d, Item: %s] ", q->orders[(q->front + i) % MAX_SIZE].customerId, q->orders[(q->front + i) % MAX_SIZE].item);
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 101, "Laptop");
    enqueue(&q, 102, "Smartphone");
    enqueue(&q, 103, "Headphones");

    displayQueue(&q);

    Order order = dequeue(&q);
    printf("Processing order: Customer ID: %d, Item: %s\n", order.customerId, order.item);
    displayQueue(&q);

    return 0;
}
```

14.public transport sheduling

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

```
#include <stdlib.h>

typedef struct Node {
    int busId;
    int priority; // 1 for express, 0 for normal
    struct Node *next;
} Node;

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

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

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

void enqueue(Queue *q, int busId, int priority) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->busId = busId;
    newNode->priority = priority;
    newNode->next = NULL;
    if (priority == 1) { // Express bus gets priority
        newNode->next = q->front;
        q->front = newNode;
        if (q->rear == NULL) {
            q->rear = newNode;
        }
    } else {
        if (isEmpty(q)) {
            q->front = q->rear = newNode;
        } else {
            q->rear->next = newNode;
            q->rear = newNode;
        }
    }
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No buses in the queue.\n");
        return -1;
    }
    int busId = q->front->busId;
    Node *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return busId;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No buses in the queue.\n");
        return;
    }
    Node *current = q->front;
    printf("Buses in queue: ");
    while (current) {
        printf("Bus ID: %d%s ", current->busId, current->priority ? " (Express)" : "");
        current = current->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 101, 0); // Normal bus
    enqueue(&q, 102, 1); // Express bus
    enqueue(&q, 103, 0); // Normal bus

    displayQueue(&q);

    printf("Bus departing: %d\n", dequeue(&q));
    displayQueue(&q);
    return 0;
}
```

15.political rally crowd control

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

#define MAX_SIZE 100

typedef struct {
    int personId;
    char type[20]; // VIP or Regular
} Person;

typedef struct {
    Person persons[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}
```

```
int isEmpty(Queue *q) {
    return q->size == 0;
}

void enqueue(Queue *q, int personId, char *type) {
    if (isFull(q)) {
        printf("Queue is full. Cannot add more people.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->persons[q->rear].personId = personId;
    strcpy(q->persons[q->rear].type, type);
    q->size++;
}

Person dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No people in the queue.\n");
        Person emptyPerson = {-1, ""};
        return emptyPerson;
    }
    Person person = q->persons[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return person;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No people in the queue.\n");
        return;
    }
    printf("Crowd in queue: ");
    for (int i = 0; i < q->size; i++) {
        printf("[Person ID: %d, Type: %s] ", q->persons[(q->front + i) % MAX_SIZE].personId, q->persons[(q-
>front + i) % MAX_SIZE].type);
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 101, "Regular");
    enqueue(&q, 102, "VIP");
    enqueue(&q, 103, "Regular");

    displayQueue(&q);

    Person person = dequeue(&q);
    printf("Processing entry for: Person ID: %d, Type: %s\n", person.personId, person.type);
    displayQueue(&q);

    return 0;
}
```

16.finalncial transaction processing

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

typedef struct Node {
    int transactionId;
    char type[10]; // Deposit or Withdrawal
    int amount;
    struct Node *next;
} Node;

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

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

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

void enqueue(Queue *q, int transactionId, char *type, int amount) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->transactionId = transactionId;
    strcpy(newNode->type, type);
    newNode->amount = amount;
    newNode->next = NULL;

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

Node *dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No transactions to process.\n");
        return NULL;
    }
    Node *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    return temp;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No transactions in the queue.\n");
        return;
    }
    Node *current = q->front;
    printf("Transactions in queue: ");
    while (current) {
        printf("[Transaction ID: %d, Type: %s, Amount: %d] ", current->transactionId, current->type, current-
>amount);
        current = current->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 1001, "Deposit", 500);
    enqueue(&q, 1002, "Withdrawal", 200);
    enqueue(&q, 1003, "Deposit", 1000);

    displayQueue(&q);

    Node *transaction = dequeue(&q);
    printf("Processing transaction: ID: %d, Type: %s, Amount: %d\n", transaction->transactionId,
transaction->type, transaction->amount);
    displayQueue(&q);

    return 0;
}
```

17.election polling booth management

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

#define MAX_SIZE 100

typedef struct {
    int voterId;
    char name[50];
} Voter;

typedef struct {
    Voter voters[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}

int isEmpty(Queue *q) {
    return q->size == 0;
```

```
}

void enqueue(Queue *q, int voterId, char *name) {
    if (isFull(q)) {
        printf("Queue is full. Cannot add more voters.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->voters[q->rear].voterId = voterId;
    strcpy(q->voters[q->rear].name, name);
    q->size++;
}

Voter dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No voters in the queue.\n");
        Voter emptyVoter = {-1, ""};
        return emptyVoter;
    }
    Voter voter = q->voters[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return voter;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No voters in the queue.\n");
        return;
    }
    printf("Voters in queue: ");
    for (int i = 0; i < q->size; i++) {
        printf("[Voter ID: %d, Name: %s] ", q->voters[(q->front + i) % MAX_SIZE].voterId, q->voters[(q->front
+ i) % MAX_SIZE].name);
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 101, "Alice");
    enqueue(&q, 102, "Bob");
    enqueue(&q, 103, "Charlie");

    displayQueue(&q);

    Voter voter = dequeue(&q);
    printf("Processing voter: Voter ID: %d, Name: %s\n", voter.voterId, voter.name);
    displayQueue(&q);

    return 0;
}
```

18.Hospital Emergency room queue

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

```
typedef struct Node {
    int patientId;
    int severity; // 1 for severe, 0 for normal
    struct Node *next;
} Node;

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

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

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

void enqueue(Queue *q, int patientId, int severity) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->patientId = patientId;
    newNode->severity = severity;
    newNode->next = NULL;
    if (severity == 1) { // Severe patients get priority
        newNode->next = q->front;
        q->front = newNode;
        if (q->rear == NULL) {
            q->rear = newNode;
        }
    } else {
        if (isEmpty(q)) {
            q->front = q->rear = newNode;
        } else {
            q->rear->next = newNode;
            q->rear = newNode;
        }
    }
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No patients in the queue.\n");
        return -1;
    }
    int patientId = q->front->patientId;
    Node *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return patientId;
```

```
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No patients in the queue.\n");
        return;
    }
    Node *current = q->front;
    printf("Patients in queue: ");
    while (current) {
        printf("[Patient ID: %d, Severity: %d] ", current->patientId, current->severity);
        current = current->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 101, 1); // Severe
    enqueue(&q, 102, 0); // Normal
    enqueue(&q, 103, 1); // Severe

    displayQueue(&q);

    printf("Treating patient: %d\n", dequeue(&q));
    displayQueue(&q);
    return 0;
}
```

19.political survey data colection

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

#define MAX_SIZE 100

typedef struct {
    int surveyorId;
    int responses; // Number of responses collected
} SurveyData;

typedef struct {
    SurveyData data[MAX_SIZE];
    int front, rear, size;
} Queue;

void initializeQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}

int isFull(Queue *q) {
    return q->size == MAX_SIZE;
}
```

```
int isEmpty(Queue *q) {
    return q->size == 0;
}

void enqueue(Queue *q, int surveyorId, int responses) {
    if (isFull(q)) {
        printf("Queue is full. Cannot add more surveyors.\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->data[q->rear].surveyorId = surveyorId;
    q->data[q->rear].responses = responses;
    q->size++;
}

SurveyData dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No surveyors in the queue.\n");
        SurveyData emptyData = {-1, 0};
        return emptyData;
    }
    SurveyData data = q->data[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->size--;
    return data;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No surveyors in the queue.\n");
        return;
    }
    printf("Surveyors in queue: ");
    for (int i = 0; i < q->size; i++) {
        printf("[Surveyor ID: %d, Responses: %d] ", q->data[(q->front + i) % MAX_SIZE].surveyorId, q-
>data[(q->front + i) % MAX_SIZE].responses);
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 101, 50);
    enqueue(&q, 102, 30);
    enqueue(&q, 103, 40);

    displayQueue(&q);

    SurveyData data = dequeue(&q);
    printf("Processing surveyor: Surveyor ID: %d, Responses: %d\n", data.surveyorId, data.responses);
    displayQueue(&q);

    return 0;
}
```

20.financial market data analysis

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

typedef struct Node {
    int dataPoint;
    struct Node *next;
} Node;

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

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

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

void enqueue(Queue *q, int dataPoint) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->dataPoint = dataPoint;
    newNode->next = NULL;
    if (isEmpty(q)) {
        q->front = q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}

int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("No data in the queue.\n");
        return -1;
    }
    int dataPoint = q->front->dataPoint;
    Node *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return dataPoint;
}

void displayQueue(Queue *q) {
    if (isEmpty(q)) {
        printf("No data in the queue.\n");
        return;
    }
    Node *current = q->front;
    printf("Market data in queue: ");
    while (current) {
        printf("[Data Point: %d] ", current->dataPoint);
        current = current->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 1500);
    enqueue(&q, 1550);
    enqueue(&q, 1600);

    displayQueue(&q);

    printf("Analyzing data: %d\n", dequeue(&q));
    displayQueue(&q);

    return 0;
}
```