```c
/*Stock Market Order Matching System**: Implement a queue using arrays to
simulate a stock market's order matching system.
 Design a program where buy and sell orders are placed in a queue.
The system should match and process orders based on price and time
priority.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int front;
    int rear;
    int size;
    int *Q;
};
void create(struct queue *q,int n)
{
    q->Q=(int *)malloc(n*sizeof(int));
}
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter price : ");
        scanf("%d",&q->Q[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
    }
}
void display(struct queue q)
{
```

```
    for(int i=q.front+1;i<=q.rear;i++)
    printf("%d -> ",q.Q[i]);
    printf("\n");
}
void main()
{
    struct queue q;
    int c;
    printf("Enter size of queue : ");
    scanf("%d",&q.size);
    q.front=-1;
    q.rear=-1;
    do
    {printf("1.Create\n");
    printf("2.Enqueue\n");
    printf("3.Dequeue\n");
    printf("4.Display\n");
    printf("5.Exit\n");
    printf("Enter choice : ");
    scanf("%d",&c);
        switch(c)
        {
            case 1:create(&q,q.size);
            break;
            case 2:enqueue(&q);
            break;
            case 3:dequeue(&q);
            break;
            case 4:display(q);
            break;
            case 5:printf("Exiting....\n");
            break;
        }
    } while (c!=5);


}
```

```
5.Exit
Enter choice : 4
50 -> 10 -> 60 ->
1.Create
2.Enqueue
3.Dequeue
4.Display
5.Exit
Enter choice : 3
1.Create
2.Enqueue
3.Dequeue
4.Display
5.Exit
Enter choice : 4
10 -> 60 ->
1.Create
2.Enqueue
3.Dequeue
4.Display
5.Exit
Enter choice :
```

```
/*Customer Service Center Simulation**: Use a linked list to implement a queue for a customer service center.
Each customer has a priority level based on their membership status, and the program should handle priority-based
queueing and dynamic customer arrival*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int priority;
    char status[10];
    struct queue *next;
}*front=NULL,*rear=NULL;
void enqueue()
{
    struct queue *new=(struct queue *)malloc(sizeof(struct queue));
```

```
    if(new==NULL)
    printf("Queue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter priority : ");
        scanf("%d",&new->priority);
        printf("Enter status : ");
        scanf("%s",new->status);
        if(front ==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct queue *t;
    if(front==NULL)
    printf("Queue is eempty\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct queue *ptr=front;
    while(ptr!=NULL)
    {
        printf("Priority :%d | Status : %s \n",ptr->priority,ptr->status);
        ptr=ptr->next;
    }
}
void main()
```

```
{
    int c;
    do
    {
    printf("1.Enqueue\n");
    printf("2.Dequeue\n");
    printf("3.Display\n");
    printf("4.Exit\n");
    printf("Enter choice : ");
    scanf("%d",&c);
        switch(c)
        {
            case 1:enqueue();
            break;
            case 2:dequeue();
            break;
            case 3:display();
            break;
            case 4:printf("Exiting....\n");
            break;
        }
    } while (c!=4);


}
```

```
 4.Exit
 Enter choice : 1
 Enter priority : 2
 Enter status : off
 1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
 Enter choice : 3
 Priority :4 | Status : on
 Priority :2 | Status : off
 1.Enqueue
 2.Dequeue
 3.Display
 4.Exit
```

```c
/*Implement a queue using arrays to manage attendees at a political
campaign event.
 The system should handle registration, check-in, and priority access for
VIP attendees.*/
 #include<stdio.h>
 #include<stdlib.h>
 struct queue
 {
    int front;
    int rear;
    int size;
    int *p;
 };
 void enqueue(struct queue *q)
 {
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter the priority : ");
        scanf("%d",&q->p[q->rear]);
    }
 }
```

```
void dequeue(struct queue *q)
{
   if(q->front==q->rear)
   printf("Queue is empty\n");
   else
   {
       q->front++;
   }
}
void display(struct queue q)
{
   for(int i=q.front+1;i<=q.rear;i++)
   {
       printf("Priority : %d\n",q.p[i]);
   }
}
void main()
{
   struct queue q;
   int c;
   printf("Enter size of queue : ");
   scanf("%d",&q.size);
   q.front=-1;
   q.rear=-1;
   q.p=(int *)malloc(q.size*sizeof(int));
   do
   {
   printf("1.Enqueue\n");
   printf("2.Dequeue\n");
   printf("3.Display\n");
   printf("4.Exit\n");
   printf("Enter choice : ");
   scanf("%d",&c);
       switch(c)
       {
           case 1:enqueue(&q);
           break;
           case 2:dequeue(&q);
           break;
           case 3:display(q);
```

```
            break;
            case 4:printf("Exiting....\n");
            break;
        }
    } while (c!=4);


}
```

```
3.Display
4.Exit
Enter choice : 3
Priority : 3
Priority : 2
Priority : 5
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice :
```

```
/*Develop a program using a linked list to simulate a queue at a bank.
Customers arrive at random intervals,
and each teller can handle one customer at a time.
The program should simulate multiple tellers and different transaction
times.*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#define MAX_TELLERS 3
#define MAX_TRANSACTION_TIME 5
struct Customer {
    int id;
    int transaction_time;
    struct Customer *next;
};

struct Queue {
    struct Customer *front, *rear;
} queue = {NULL, NULL};
```

```c
int customer_count = 0;

void enqueue() {
    struct Customer *new_customer = (struct Customer
*)malloc(sizeof(struct Customer));
    new_customer->id = ++customer_count;
    new_customer->transaction_time = (rand() % MAX_TRANSACTION_TIME) + 1;
    new_customer->next = NULL;

    if (queue.rear == NULL) {
        queue.front = queue.rear = new_customer;
    } else {
        queue.rear->next = new_customer;
        queue.rear = new_customer;
    }
    printf("Customer %d arrives (Transaction Time: %d sec)\n",
new_customer->id, new_customer->transaction_time);
}

// Function to dequeue a customer
struct Customer *dequeue() {
    if (queue.front == NULL) return NULL;

    struct Customer *temp = queue.front;
    queue.front = queue.front->next;
    if (queue.front == NULL) queue.rear = NULL;
    return temp;
}
void process_customers() {
    int teller_busy[MAX_TELLERS] = {0};
    struct Customer *teller_queue[MAX_TELLERS] = {NULL};

    while (queue.front != NULL || teller_busy[0] || teller_busy[1] ||
teller_busy[2]) {
        for (int i = 0; i < MAX_TELLERS; i++) {
            if (!teller_busy[i]) {
                teller_queue[i] = dequeue();
                if (teller_queue[i]) {
                    teller_busy[i] = teller_queue[i]->transaction_time;
```

```
                    printf("Teller %d starts processing Customer %d for %d
seconds.\n", i + 1, teller_queue[i]->id, teller_busy[i]);
                }
            }
        }

        sleep(1);

        for (int i = 0; i < MAX_TELLERS; i++) {
            if (teller_busy[i]) {
                teller_busy[i]--;
                if (teller_busy[i] == 0) {
                    printf("Teller %d finished processing Customer %d.\n",
i + 1, teller_queue[i]->id);
                    free(teller_queue[i]);
                    teller_queue[i] = NULL;
                }
            }
        }

        if (rand() % 3 == 0) {
            enqueue();
        }
    }
    printf("All customers have been served.\n");
}

int main() {
    srand(time(NULL));
    printf("Bank queue simulation started...\n");

    for (int i = 0; i < 5; i++) {
        enqueue();
    }

    process_customers();
    return 0;
}
```

```
Customer 1 arrives (Transaction Time: 3 sec)
Customer 2 arrives (Transaction Time: 1 sec)
Customer 3 arrives (Transaction Time: 3 sec)
Customer 4 arrives (Transaction Time: 3 sec)
Customer 5 arrives (Transaction Time: 2 sec)
Teller 1 starts processing Customer 1 for 3 seconds.
Teller 2 starts processing Customer 2 for 1 seconds.
Teller 3 starts processing Customer 3 for 3 seconds.
Teller 2 finished processing Customer 2.
Teller 2 starts processing Customer 4 for 3 seconds.
Customer 6 arrives (Transaction Time: 5 sec)
Teller 1 finished processing Customer 1.
Teller 3 finished processing Customer 3.
Customer 7 arrives (Transaction Time: 5 sec)
Teller 1 starts processing Customer 5 for 2 seconds.
Teller 3 starts processing Customer 6 for 5 seconds.
Teller 2 finished processing Customer 4.
Customer 8 arrives (Transaction Time: 5 sec)
Teller 2 starts processing Customer 7 for 5 seconds.
Teller 1 finished processing Customer 5.
Teller 1 starts processing Customer 8 for 5 seconds.
PS D:\projects\quest\C>
```

```c
/*Implement a queue using arrays to process real-time data feeds from
multiple financial instruments.
The system should handle high-frequency data inputs and ensure data
integrity and order.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int size;
    int front;
    int rear;
    int *data;
};
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
```

```
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter data : ");
        scanf("%d",&q->data[q->rear]);
    }
}
void dequeue(struct queue *q)
{
    if(q->rear == q->front)
    {
        q->front++;
    }
}
void display(struct queue q)
{
    for(int i=q.front+1;i<=q.rear;i++)
    printf("data : %d|",q.data[i]);
    printf("\n");
}
 void main()
{
    struct queue q;
    int c;
    printf("Enter size of queue : ");
    scanf("%d",&q.size);
    q.front=-1;
    q.rear=-1;
    q.data=(int *)malloc(q.size*sizeof(int));
    do
    {
    printf("1.Enqueue\n");
    printf("2.Dequeue\n");
    printf("3.Display\n");
    printf("4.Exit\n");
    printf("Enter choice : ");
    scanf("%d",&c);
        switch(c)
        {
```

```
            case 1:enqueue(&q);
            break;
            case 2:dequeue(&q);
            break;
            case 3:display(q);
            break;
            case 4:printf("Exiting....\n");
            break;
        }
    } while (c!=4);


}
```

```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice : 3
data : 300|data : 400|data : 500|
1.Enqueue
2.Dequeue
3.Display
```

```
/*Use a linked list to implement a queue for cars at a traffic light.
The system should manage cars arriving at different times and simulate the
light changing from red to green.*/
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

const int max=2;
struct queue {
    int car;
    struct queue *next;
}*front=NULL,*rear=NULL;
static int m=0;

void enqueue() {
    int n;
    n = rand() % 10;
    for (int i = 0; i < n; i++) {
```

```
        struct queue *new = (struct queue *)malloc(sizeof(struct queue));
        if (new == NULL) {
            printf("Queue is full\n");
            return;
        }
        new->next = NULL;
        new->car = m++;

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

void dequeue() {
    int n;
    n = rand() % 10 + 1;
    struct queue *t;

    for (int i = 0; i < n; i++) {
        if (front == NULL) {
            printf("Queue is empty\n");
            return;
        } else {
            t = front;
            front = front->next;
            free(t);
        }
    }
}

void display() {
    struct queue *ptr = front;
    while (ptr != NULL) {
        printf("Car no : %d \n", ptr->car);
        ptr = ptr->next;
    }
```

```
}

int main() {
    int c, t;
    srand(time(0));

    printf("Enter number of cycles: ");
    scanf("%d", &t);

    while (t != 0) {
        c = rand() % 2;

        if (c == 1) {
            enqueue();
            printf("Car after enqueue\n");
            display();
        } else {
            dequeue();
            printf("Car after dequeue\n");
            display();
        }

        t--;
    }

    return 0;
}
```

```
Enter number of cycles: 5
Car after enqueue
Car no : 0
Car after enqueue
Car no : 0
Car no : 1
Queue is empty
Car after dequeue
Car after enqueue
Car no : 2
Car no : 3
Car no : 4
Car no : 5
Car no : 6
Car no : 7
Car no : 8
Car no : 9
Car no : 10
Car after dequeue
Car no : 3
Car no : 4
```

```c
/*Implement a queue using arrays to manage the vote counting process
during an election.
The system should handle multiple polling stations and ensure votes are
counted in the order received.*/
#include <stdio.h>
#include <stdlib.h>

#define MAX_VOTES 100
#define MAX_CANDIDATES 5

struct Queue {
    int votes[MAX_VOTES];
    int front, rear;
} voteQueue;
void initQueue() {
    voteQueue.front = -1;
    voteQueue.rear = -1;
```

```
}
int isFull() {
    return voteQueue.rear == MAX_VOTES - 1;
}
int isEmpty() {
    return voteQueue.front == -1 || voteQueue.front > voteQueue.rear;
}
void enqueue(int candidate) {
    if (isFull()) {
        printf("Vote queue is full! Cannot accept more votes.\n");
        return;
    }
    if (voteQueue.front == -1) voteQueue.front = 0;
    voteQueue.rear++;
    voteQueue.votes[voteQueue.rear] = candidate;
    printf("Vote for candidate %d recorded.\n", candidate);
}
int dequeue() {
    if (isEmpty()) {
        printf("No votes left to count!\n");
        return -1;
    }
    return voteQueue.votes[voteQueue.front++];
}
void countVotes() {
    int voteCounts[MAX_CANDIDATES] = {0};
    while (!isEmpty()) {
        int candidate = dequeue();
        if (candidate >= 1 && candidate <= MAX_CANDIDATES) {
            voteCounts[candidate - 1]++;
        }
    }
    printf("Vote count results:\n");
    for (int i = 0; i < MAX_CANDIDATES; i++) {
        printf("Candidate %d: %d votes\n", i + 1, voteCounts[i]);
    }
}

int main() {
    initQueue();
```

```c
    int choice, vote;
    do {
        printf("\nElection Voting System\n");
        printf("1. Cast Vote\n");
        printf("2. Count Votes\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter candidate number (1-%d): ", MAX_CANDIDATES);
                scanf("%d", &vote);
                enqueue(vote);
                break;
            case 2:
                countVotes();
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 3);

    return 0;
}
```

```
3. Exit
Enter your choice: 1
Enter candidate number (1-5): 3
Vote for candidate 3 recorded.

Election Voting System
1. Cast Vote
2. Count Votes
3. Exit
Enter your choice: 2
Vote count results:
Candidate 1: 1 votes
Candidate 2: 1 votes
Candidate 3: 1 votes
Candidate 4: 0 votes
Candidate 5: 0 votes

Election Voting System
1. Cast Vote
2. Count Votes
3. Exit
Enter your choice:
```

```c
/*Use a linked list to implement a queue for airplanes waiting to land or
take off.
The system should handle priority for emergency landings and manage runway
allocation efficiently.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    int plane;
    int emergency;
    struct queue *next;
}*front=NULL,*rear=NULL;
void enqueue()
{
    struct queue *new=(struct queue *)malloc(sizeof(struct queue));
    if(new==NULL)
```

```
    printf("Queue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter plane number : ");
        scanf("%d",&new->plane);
        printf("Emergency landing (1.yes 2.No) : ");
        scanf("%d",&new->emergency);
        if(front==NULL)
        {
            front=rear=new;
        }
        else
        {
            rear->next=new;
            rear=rear->next;

        }
    }
}
void dequeue()
{
    struct queue *t=front,*ptr=front;
    if(front==rear)
    printf("Queue is empty\n");
    else
    {
        while(ptr!=NULL)
        {
            if(ptr->emergency==1)
            break;
            ptr=ptr->next;
        }
        if(ptr==front)
        {
            front=front->next;
            free(ptr);
        }
        else
        {
```

```
            while(t->next!=ptr)
            t=t->next;
            t->next=ptr->next;
            free(ptr);
        }
    }
}
void display()
{
    struct queue *ptr=front;
    while(ptr!=NULL)
    {
        printf("Plane id : %d | Emergency landing :
%d\n",ptr->plane,ptr->emergency);
        ptr=ptr->next;
    }
    printf("\n");
}
void main()
{
    int c;
    do
    {
        printf("1.Enqueue\n");
        printf("2.Dequeue\n");
        printf("3.Display\n");
        printf("Enter choice : ");
        scanf("%d",&c);
        switch(c)
        {
            case 1:
            enqueue();
            break;
            case 2:
            dequeue();
            break;
            case 3:
            display();
            break;
            case 4:
```

```
            printf("Exiting....\n");
            break;
        }
    } while (c!=4);

}
```

```
2.Dequeue
3.Display
Enter choice : 3
Plane id : 1002 | Emergency landing : 2
Plane id : 1003 | Emergency landing : 2
Plane id : 1004 | Emergency landing : 1

1.Enqueue
2.Dequeue
3.Display
Enter choice : 2
1.Enqueue
2.Dequeue
3.Display
Enter choice : 3
Plane id : 1002 | Emergency landing : 2
Plane id : 1003 | Emergency landing : 2

1.Enqueue
2.Dequeue
3.Display
Enter choice :
```

```
/*Develop a program using arrays to simulate a queue for stock trading
orders.
The system should manage buy and sell orders, handle order cancellations,
and provide real-time updates.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_ORDERS 10
```

```c
typedef struct {
    int order_id;
    char type[5];
    int quantity;
    float price;
} Order;

typedef struct {
    Order orders[MAX_ORDERS];
    int front, rear, count;
} OrderQueue;

OrderQueue queue = {.front = 0, .rear = -1, .count = 0};
int order_counter = 1;

int is_full() {
    return queue.count == MAX_ORDERS;
}

int is_empty() {
    return queue.count == 0;
}

void enqueue_order() {
    if (is_full()) {
        printf("Order queue is full. Cannot place more orders.\n");
        return;
    }

    Order new_order;
    new_order.order_id = order_counter++;

    printf("Enter order type (BUY/SELL): ");
    scanf("%s", new_order.type);
    printf("Enter quantity: ");
    scanf("%d", &new_order.quantity);
    printf("Enter price: ");
    scanf("%f", &new_order.price);

    queue.rear = (queue.rear + 1) % MAX_ORDERS;
```

```
    queue.orders[queue.rear] = new_order;
    queue.count++;

    printf("Order placed: ID=%d, Type=%s, Quantity=%d, Price=%.2f\n",
           new_order.order_id, new_order.type, new_order.quantity,
new_order.price);
}

void process_order() {
    if (is_empty()) {
        printf("No orders to process.\n");
        return;
    }

    Order processed_order = queue.orders[queue.front];
    queue.front = (queue.front + 1) % MAX_ORDERS;
    queue.count--;

    printf("Processing Order: ID=%d, Type=%s, Quantity=%d, Price=%.2f\n",
           processed_order.order_id, processed_order.type,
processed_order.quantity, processed_order.price);
}

void cancel_order() {
    if (is_empty()) {
        printf("No orders available to cancel.\n");
        return;
    }

    int order_id;
    printf("Enter Order ID to cancel: ");
    scanf("%d", &order_id);

    int i, index = -1;
    for (i = 0; i < queue.count; i++) {
        int pos = (queue.front + i) % MAX_ORDERS;
        if (queue.orders[pos].order_id == order_id) {
            index = pos;
            break;
        }
```

```
    }

    if (index == -1) {
        printf("Order ID %d not found in the queue.\n", order_id);
        return;
    }

    for (i = index; i != queue.rear; i = (i + 1) % MAX_ORDERS) {
        queue.orders[i] = queue.orders[(i + 1) % MAX_ORDERS];
    }

    queue.rear = (queue.rear - 1 + MAX_ORDERS) % MAX_ORDERS;
    queue.count--;

    printf("Order ID %d has been canceled.\n", order_id);
}

void display_orders() {
    if (is_empty()) {
        printf("No pending orders.\n");
        return;
    }

    printf("\nCurrent Orders in Queue:\n");
    printf("----------------------------------------\n");
    printf("ID     Type    Quantity    Price\n");
    printf("----------------------------------------\n");

    for (int i = 0; i < queue.count; i++) {
        int pos = (queue.front + i) % MAX_ORDERS;
        printf("%-6d %-6s %-10d %.2f\n",
               queue.orders[pos].order_id, queue.orders[pos].type,
               queue.orders[pos].quantity, queue.orders[pos].price);
    }
    printf("----------------------------------------\n");
}

int main() {
    int choice;
    do {
```

```
        printf("\nStock Trading Order System\n");
        printf("1. Place Order (BUY/SELL)\n");
        printf("2. Process Order\n");
        printf("3. Cancel Order\n");
        printf("4. View Orders\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: enqueue_order();
            break;
            case 2: process_order();
            break;
            case 3: cancel_order();
            break;
            case 4: display_orders();
            break;
            case 5: printf("Exiting...\n");
            break;
            default: printf("Invalid choice. Try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

```
1. Place Order (BUY/SELL)
2. Process Order
3. Cancel Order
4. View Orders
5. Exit
Enter your choice: 4

Current Orders in Queue:
----------------------------------------
ID      Type    Quantity    Price
----------------------------------------
3       SELL    10          2000.00
----------------------------------------

Stock Trading Order System
1. Place Order (BUY/SELL)
2. Process Order
3. Cancel Order
```

```c
/*Conference Registration System**: Implement a queue using linked lists
for managing registrations at a conference.
The system should handle walk-in registrations, pre-registrations, and
cancellations.*/
#include<stdio.h>
#include<stdlib.h>
struct queue
{
    char type[10];
    char name[10];
    int id;
    struct queue *next;
}*front=NULL,*rear=NULL;
void enqueue()
{
    struct queue *new=(struct queue *)malloc(sizeof(struct queue));
    if(new==NULL)
    printf("Queue is full\n");
    else
    {
        new->next=NULL;
```

```
        printf("Enter registration type : ");
        scanf("%s",new->type);
        printf("Enter name : ");
        scanf("%s",new->name);
        printf("Enter registration id : ");
        scanf("%d",&new->id);
        if(front==NULL)
        front=rear=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
    struct queue *ptr=front,*t=front;
    int id,f;
    if(front==rear)
    printf("Queue is empty\n");
    else
    {
        printf("Enter registration id : ");
        scanf("%d",&id);
        while(ptr!=NULL)
        {
            if(ptr->id==id)
            break;
            ptr=ptr->next;
        }
        if(ptr==front)
        {
            front=front->next;
            free(ptr);
        }
        else
        {
            while(t->next!=ptr)
            t=t->next;
```

```
            t->next=ptr->next;
            free(ptr);

        }

    }
}
void display()
{
    struct queue *ptr=front;
    while(ptr!=NULL)
    {
        printf("Registered name : %s | Type : %s | ID : %d
\n",ptr->name,ptr->type,ptr->id);
        ptr=ptr->next;
    }
}
void main()
{
     int c;
    do
    {
        printf("1.Enqueue\n");
        printf("2.Dequeue\n");
        printf("3.Display\n");
        printf("Enter choice : ");
        scanf("%d",&c);
        switch(c)
        {
            case 1:
            enqueue();
            break;
            case 2:
            dequeue();
            break;
            case 3:
            display();
            break;
            case 4:
            printf("Exiting....\n");
```

```
            break;
        }
    } while (c!=4);
}
```

```
Registered name : green | Type : 2 | ID : 3
1.Enqueue
2.Dequeue
3.Display
Enter choice : 2
Enter registration id : 1
1.Enqueue
2.Dequeue
3.Display
Enter choice : 3
Registered name : blu | Type : walkin | ID : 1001
Registered name : green | Type : 2 | ID : 3
1.Enqueue
2.Dequeue
3.Display
Enter choice : 2
Enter registration id : 3
1.Enqueue
2.Dequeue
3.Display
Enter choice : 2
Enter registration id :
```

```
/*Use arrays to implement a queue for managing the audience at a political debate.
The system should handle entry, seating arrangements, and priority access for media personnel.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_AUDIENCE 10

typedef struct {
    char name[50];
    char category[10];
```

```c
} Audience;

typedef struct {
    Audience queue[MAX_AUDIENCE];
    int front, rear, count;
} AudienceQueue;

AudienceQueue audience_queue = {.front = 0, .rear = -1, .count = 0};
AudienceQueue media_queue = {.front = 0, .rear = -1, .count = 0};
int is_full(AudienceQueue *q) {
    return q->count == MAX_AUDIENCE;
}

int is_empty(AudienceQueue *q) {
    return q->count == 0;
}


void enqueue() {
    if (audience_queue.count + media_queue.count == MAX_AUDIENCE) {
        printf("Audience queue is full. No more entries allowed.\n");
        return;
    }

    Audience new_audience;
    printf("Enter Name: ");
    scanf("%s", new_audience.name);
    printf("Category (MEDIA/GUEST): ");
    scanf("%s", new_audience.category);

    if (strcmp(new_audience.category, "MEDIA") == 0) {
        if (is_full(&media_queue)) {
            printf("Media queue is full. Cannot add more media
personnel.\n");
            return;
        }
        media_queue.rear = (media_queue.rear + 1) % MAX_AUDIENCE;
        media_queue.queue[media_queue.rear] = new_audience;
        media_queue.count++;
```

```
        printf("%s (MEDIA) added to priority queue.\n",
new_audience.name);
    } else {
        if (is_full(&audience_queue)) {
            printf("General audience queue is full. Cannot add more
guests.\n");
            return;
        }
        audience_queue.rear = (audience_queue.rear + 1) % MAX_AUDIENCE;
        audience_queue.queue[audience_queue.rear] = new_audience;
        audience_queue.count++;
        printf("%s (GUEST) added to general queue.\n", new_audience.name);
    }
}

void dequeue() {
    if (!is_empty(&media_queue)) {
        Audience person = media_queue.queue[media_queue.front];
        media_queue.front = (media_queue.front + 1) % MAX_AUDIENCE;
        media_queue.count--;
        printf("Seating %s (MEDIA) in priority seating.\n", person.name);
    } else if (!is_empty(&audience_queue)) {
        Audience person = audience_queue.queue[audience_queue.front];
        audience_queue.front = (audience_queue.front + 1) % MAX_AUDIENCE;
        audience_queue.count--;
        printf("Seating %s (GUEST) in general seating.\n", person.name);
    } else {
        printf("No audience members to seat.\n");
    }
}


void display_queue() {
    if (is_empty(&media_queue) && is_empty(&audience_queue)) {
        printf("No one is waiting in the queue.\n");
        return;
    }

    printf("\n*** Current Queue ***\n");
```

```c
    if (!is_empty(&media_queue)) {
        printf("Priority Queue (MEDIA):\n");
        for (int i = 0; i < media_queue.count; i++) {
            int pos = (media_queue.front + i) % MAX_AUDIENCE;
            printf(" - %s (MEDIA)\n", media_queue.queue[pos].name);
        }
    }

    if (!is_empty(&audience_queue)) {
        printf("General Queue (GUEST):\n");
        for (int i = 0; i < audience_queue.count; i++) {
            int pos = (audience_queue.front + i) % MAX_AUDIENCE;
            printf(" - %s (GUEST)\n", audience_queue.queue[pos].name);
        }
    }
    printf("\n");
}


int main() {
    int choice;
    do {
        printf("\nPolitical Debate Audience Management\n");
        printf("1. Enter Audience (Media/Guest)\n");
        printf("2. Seat Next Audience Member\n");
        printf("3. View Waiting Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: enqueue(); break;
            case 2: dequeue(); break;
            case 3: display_queue(); break;
            case 4: printf("Exiting system...\n"); break;
            default: printf("Invalid choice. Try again.\n");
        }
    } while (choice != 4);

    return 0;
```

```
}
```

```
4. Exit
Enter your choice: 2
Seating red (MEDIA) in priority seating.

Political Debate Audience Management
1. Enter Audience (Media/Guest)
2. Seat Next Audience Member
3. View Waiting Queue
4. Exit
Enter your choice: 3

*** Current Queue ***
General Queue (GUEST):
 - blu (GUEST)


Political Debate Audience Management
1. Enter Audience (Media/Guest)
2. Seat Next Audience Member
3. View Waiting Queue
4. Exit
Enter your choice:
```

```
/*Develop a queue using linked lists to manage loan applications at a bank.
The system should prioritize applications based on the loan amount and
applicant's credit score.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct LoanApplication {
    char name[50];
    float loan_amount;
    int credit_score;
    struct LoanApplication *next;
} LoanApplication;
```

```
LoanApplication *front = NULL;

LoanApplication* create_application(char *name, float loan_amount, int
credit_score) {
    LoanApplication *new_app =
(LoanApplication*)malloc(sizeof(LoanApplication));
    strcpy(new_app->name, name);
    new_app->loan_amount = loan_amount;
    new_app->credit_score = credit_score;
    new_app->next = NULL;
    return new_app;
}


void enqueue(char *name, float loan_amount, int credit_score) {
    LoanApplication *new_app = create_application(name, loan_amount,
credit_score);

    if (front == NULL ||
       (loan_amount > front->loan_amount) ||
       (loan_amount == front->loan_amount && credit_score >
front->credit_score)) {
        new_app->next = front;
        front = new_app;
        return;
    }

    LoanApplication *current = front;
    while (current->next != NULL &&
           (current->next->loan_amount > loan_amount ||
           (current->next->loan_amount == loan_amount &&
current->next->credit_score >= credit_score))) {
        current = current->next;
    }

    new_app->next = current->next;
    current->next = new_app;
}
```

```c
void dequeue() {
    if (front == NULL) {
        printf("No loan applications to process.\n");
        return;
    }

    LoanApplication *temp = front;
    printf("\nProcessing Loan Application:\n");
    printf("Applicant: %s | Loan Amount: $%.2f | Credit Score: %d\n",
temp->name, temp->loan_amount, temp->credit_score);
    front = front->next;
    free(temp);
}

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

    printf("\nLoan Applications Queue (Priority Order):\n");
    LoanApplication *current = front;
    while (current != NULL) {
        printf("Applicant: %s | Loan Amount: $%.2f | Credit Score: %d\n",
               current->name, current->loan_amount,
current->credit_score);
        current = current->next;
    }
}


int main() {
    int choice;
    char name[50];
    float loan_amount;
    int credit_score;

    do {
        printf("\nBank Loan Application System\n");
```

```
        printf("1. Submit Loan Application\n");
        printf("2. Process Next Application\n");
        printf("3. View Pending Applications\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        getchar();

        switch (choice) {
            case 1:
                printf("Enter Applicant Name: ");
                fgets(name, sizeof(name), stdin);
                name[strcspn(name, "\n")] = 0;
                printf("Enter Loan Amount: $");
                scanf("%f", &loan_amount);
                printf("Enter Credit Score (300-850): ");
                scanf("%d", &credit_score);
                enqueue(name, loan_amount, credit_score);
                printf("Loan application submitted successfully.\n");
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display_queue();
                break;
            case 4:
                printf("Exiting system...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

```
an Applications Queue (Priority Order):
plicant: blu | Loan Amount: $40000.00 | Credit Score: 600
plicant: red | Loan Amount: $2000.00 | Credit Score: 500

nk Loan Application System
 Submit Loan Application
 Process Next Application
 View Pending Applications
 Exit
ter choice: 2

ocessing Loan Application:
plicant: blu | Loan Amount: $40000.00 | Credit Score: 600

nk Loan Application System
 Submit Loan Application
 Process Next Application
 View Pending Applications
 Exit
ter choice:
```

```c
/*Implement a queue using arrays for an online shopping platform's
checkout system.
 The program should handle multiple customers checking out simultaneously
and manage inventory updates.*/
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_CUSTOMERS 10
#define MAX_ITEMS 5
typedef struct {
    char name[50];
    int cart[MAX_ITEMS];
} Customer;
typedef struct {
    Customer queue[MAX_CUSTOMERS];
    int front, rear;
} CheckoutQueue;
int inventory[MAX_ITEMS] = {10, 10, 10, 10, 10};
```

```
char *item_names[MAX_ITEMS] = {"Laptop", "Phone", "Headphones", "Tablet",
"Smartwatch"};
CheckoutQueue checkout = {.front = -1, .rear = -1};

int isFull() {
    return (checkout.rear + 1) % MAX_CUSTOMERS == checkout.front;
}

int isEmpty() {
    return checkout.front == -1;
}

void enqueue() {
    if (isFull()) {
        printf("Checkout queue is full! Please wait...\n");
        return;
    }
    Customer new_customer;
    printf("Enter customer name: ");
    scanf("%s", new_customer.name);

    printf("Enter quantity for each item:\n");
    for (int i = 0; i < MAX_ITEMS; i++) {
        printf("%s (Available: %d): ", item_names[i], inventory[i]);
        scanf("%d", &new_customer.cart[i]);
    }
    if (isEmpty()) {
        checkout.front = checkout.rear = 0;
    } else {
        checkout.rear = (checkout.rear + 1) % MAX_CUSTOMERS;
    }

    checkout.queue[checkout.rear] = new_customer;
    printf("Customer %s added to checkout queue.\n", new_customer.name);
}
void dequeue() {
    if (isEmpty()) {
        printf("No customers in checkout queue.\n");
        return;
    }
```

```
    Customer current_customer = checkout.queue[checkout.front];
    printf("\nProcessing checkout for %s...\n", current_customer.name);
    int canProcess = 1;
    for (int i = 0; i < MAX_ITEMS; i++) {
        if (current_customer.cart[i] > inventory[i]) {
            printf("Not enough %s in stock! Cannot process checkout.\n",
item_names[i]);
            canProcess = 0;
            break;
        }
    }
    if (canProcess) {
        for (int i = 0; i < MAX_ITEMS; i++) {
            inventory[i] -= current_customer.cart[i];
        }
        printf("Checkout completed for %s.\n", current_customer.name);
    }
    if (checkout.front == checkout.rear) {
        checkout.front = checkout.rear = -1;
    } else {
        checkout.front = (checkout.front + 1) % MAX_CUSTOMERS;
    }
}
void displayQueue() {
    if (isEmpty()) {
        printf("No customers in checkout queue.\n");
        return;
    }

    printf("\nCurrent Checkout Queue:\n");
    int i = checkout.front;
    while (1) {
        printf("Customer: %s\n", checkout.queue[i].name);
        if (i == checkout.rear) break;
        i = (i + 1) % MAX_CUSTOMERS;
    }
}
void displayInventory() {
    printf("\nCurrent Inventory:\n");
    for (int i = 0; i < MAX_ITEMS; i++) {
```

```
        printf("%s: %d in stock\n", item_names[i], inventory[i]);
    }
}

int main() {
    int choice;
    do {
        printf("\nOnline Shopping Checkout System\n");
        printf("1. Add Customer to Queue\n");
        printf("2. Process Checkout\n");
        printf("3. View Checkout Queue\n");
        printf("4. View Inventory\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                displayQueue();
                break;
            case 4:
                displayInventory();
                break;
            case 5:
                printf("Exiting system...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

```
Headphones: 10 in stock
Tablet: 10 in stock
Smartwatch: 10 in stock

Online Shopping Checkout System
1. Add Customer to Queue
2. Process Checkout
3. View Checkout Queue
4. View Inventory
5. Exit
Enter choice: 2

Processing checkout for red...
Checkout completed for red.

Online Shopping Checkout System
1. Add Customer to Queue
2. Process Checkout
3. View Checkout Queue
4. View Inventory
5. Exit
Enter choice:
```

```c
/*se linked lists to implement a queue for managing bus arrivals and
departures at a terminal.
The system should handle peak hours, off-peak hours, and prioritize
express buses.*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct  queue
{
    int id;
    int express;
```

```
    struct queue *next;

}*front=NULL,*rear=NULL;
void enqueue()
{
struct queue *new=(struct queue *)malloc(sizeof(struct queue));
struct queue *ptr=front;
if(new==NULL)
printf("Queue is full\n");
else
{
    new->next=NULL;
    printf("Enter bus id : ");
    scanf("%d",&new->id);
    printf("1.Express or 2.normal : ");
    scanf("%d",&new->express);
    if(front==NULL)
    {
        front=rear=new;
    }
    else
    {
        if(new->express==1)
        {
            new->next=front;
            front=new;
        }
        else
        {
            rear->next=new;
            rear=rear->next;
        }

    }

}
}
void dequeue()
{
    struct queue *t;
```

```
    if(front==rear)
    printf("Queue is empty\n");
    else
    {
        t=front;
        front=front->next;
        free(t);
    }

}
void display()
{
    struct queue *t=front;
    while(t!=NULL)
    {
        if(t->express==1)
        printf("Bus id : %d | Bus type : Express\n",t->id);
        else
        printf("Bus id : %d | Bus type : Lowfloor\n",t->id);
        t=t->next;
    }

}
void main()
{
    int c;
    do
    {
        printf("1.Enqueue\n");
        printf("2.Dequeue\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&c);
        switch(c)
        {
            case 1: enqueue();
            break;
            case 2: dequeue();
            break;
```

```
            case 3: display();
            break;
            case 4: printf("Exiting...\n");
            break;
        }
    } while (c!=4);

}
```

```
2.Dequeue
3.Display
4.Exit
Enter choice
3
Bus id : 3 | Bus type : Express
Bus id : 2 | Bus type : Express
Bus id : 1 | Bus type : Lowfloor
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice
2
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice
3
Bus id : 2 | Bus type : Express
Bus id : 1 | Bus type : Lowfloor
```

```
/*Develop a queue using arrays to manage the crowd at a political rally.
The system should handle entry, exit, and VIP sections, ensuring safety
and order.*/
#include<stdio.h>
#include<stdlib.h>
struct crowd
{
    int type;
```

```
    char name[10];
};
struct queue
{
    int size;
    int front;
    int rear;
    struct crowd *c;
};
void enqueue(struct queue *q) {
    struct crowd n, t;

    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
        return;
    }
    q->rear++;

    printf("Enter name: ");
    scanf("%s", n.name);
    printf("Enter type (1.VIP, 2.Common): ");
    scanf("%d", &n.type);
    if (n.type == 1) {
        if (q->rear == 0) {
            q->c[q->rear] = n;
        } else {

            int i;
            for (i = q->rear; i > q->front && q->c[i - 1].type == 2; i--)
{
                q->c[i] = q->c[i - 1];
            }
            q->c[i] = n;
        }
    } else {
        q->c[q->rear] = n;
    }
}
void dequeue(struct queue *q)
{
```

```
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
    }
}
void display(struct queue q)
{
    for(int i=q.front+1;i<=q.rear;i++)
    {
    if(q.c[i].type==1)
    printf("Name : %s | Type : VIP\n",q.c[i].name);
    else
    printf("Name : %s | Type : Common\n",q.c[i].name);

    }


}
void main()
{
    struct queue q;
    int choice;
    printf("Enter size of queue : ");
    scanf("%d",&q.size);
    q.front=-1;
    q.rear=-1;
    q.c=(struct crowd *)malloc(q.size*sizeof(struct crowd));
    do
    {
    printf("1.Enqueue\n");
    printf("2.Dequeue\n");
    printf("3.Display\n");
    printf("4.Exit\n");
    printf("Enter choice : ");
    scanf("%d",&choice);
        switch(choice)
        {
            case 1:enqueue(&q);
```

```
            break;
            case 2:dequeue(&q);
            break;
            case 3:display(q);
            break;
            case 4:printf("Exiting....\n");
            break;
        }
    } while (choice!=4);

}
```

```
Enter type (1 = VIP, 2 = Common): 2

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 1
Enter name: blu
Enter type (1 = VIP, 2 = Common): 1

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 3

Queue Status:
Name: blu | Type: VIP
Name: red | Type: Common

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 2
Dequeued: blu (Type: VIP)

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice: 3

Queue Status:
```

```c
/*Implement a queue using linked lists to process financial transactions.
The system should handle deposits, withdrawals, and transfers, ensuring
real-time processing and accuracy.*/
#include<stdio.h>
```

```c
#include<stdlib.h>
#include<string.h>
struct queue
{
    float amount;
    int type;//1.deposit,2.withdraw,3.transfer
    char to[10];
    struct queue *next;
}*front=NULL,*rear=NULL;
void enqueue()
{
    struct queue *new=(struct queue *)malloc(sizeof(struct queue));
    if(new==NULL)
    printf("Queue is full\n");
    else
    {
        new->next=NULL;
        printf("Enter transaction type(1.deposit,2.withdraw,3.transfer) :
");
        scanf("%d",&new->type);
        if(new->type==3)
        {
            printf("Enter to account no : ");
            scanf("%s",new->to);
        }
        else
        strcpy(new->to,"\0");
        printf("Enter amount : ");
        scanf("%f",&new->amount);
        if(rear==NULL)
        rear=front=new;
        else
        {
            rear->next=new;
            rear=rear->next;
        }
    }
}
void dequeue()
{
```

```
    struct queue *t;
    if(front==NULL)
    printf("Queue is empty\n");
    else
    {
        printf("Processing transaction\n");
        if(front->type==3)
        {
            printf("transaction type : Transfer\n");
            printf("To account : %s\n",front->to);
        }
        else
        {
            if(front->type==1)
            printf("transaction type : Deposit\n");
            else if(front->type==2)
            printf("transaction type : Withdraw\n");
        }
        printf("Amount : %f\n",front->amount);
        t=front;
        front=front->next;
        free(t);
    }
}
void display()
{
    struct queue *ptr=front;
    while(ptr!=NULL)
    {
        if(ptr->type==1)
          printf("Type : Deposit, Amount : $%.2f\n",ptr->amount);
          else if(ptr->type==2)
           printf("Type : Withdraw, Amount : $%.2f\n",ptr->amount);
           else if(ptr->type==3)
           printf("Type : Transfer,To : %s ,Amount :
$%.2f\n",ptr->to,ptr->amount);
        ptr=ptr->next;
    }

}
```

```c
void main()
{
    int c;
    do
    {
        printf("1.Enqueue\n");
        printf("2.Dequeue\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("Enter choice\n");
        scanf("%d",&c);
        switch(c)
        {
            case 1: enqueue();
            break;
            case 2: dequeue();
            break;
            case 3: display();
            break;
            case 4: printf("Exiting...\n");
            break;
        }
    } while (c!=4);
}
```

```
Enter amount : 4000
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice
1
Enter transaction type(1.deposit,2.withdraw,3.transfer) : 3
Enter to account no : 10001
Enter amount : 500
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice
3
Type : Deposit, Amount : $50000.00
Type : Withdraw, Amount : $4000.00
Type : Transfer,To : 10001 ,Amount : $500.00
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice
2
Processing transaction
transaction type : Deposit
Amount : 50000.000000
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice
```

```c
/*Use arrays to implement a queue for managing voters at a polling booth.
The system should handle voter registration, verification, and ensure
smooth voting process.*/
#include<stdio.h>
```

```c
#include<stdlib.h>
#include<string.h>
struct vote
{
    int id;
    char symbol;
};
struct queue
{
    int front;
    int size;
    int rear;
    struct vote *v;
};
void enqueue(struct queue *q)
{
    if(q->rear==q->size-1)
    printf("Queue is full\n");
    else
    {
        q->rear++;
        printf("Enter voter id : ");
        scanf("%d",&q->v[q->rear].id);
        getchar();
        printf("Enter symbol : ");
        scanf("%c",&q->v[q->rear].symbol);
    }
}
void dequeue(struct queue *q)
{
    if(q->front==q->rear)
    printf("Queue is empty\n");
    else
    {
        q->front++;
        printf("Verification process \n");
        printf("Voter id : %d\n",q->v[q->front].id);
        printf("Symbol : %c\n",q->v[q->front].symbol);
```

```
    }
}
void display(struct queue q)
{
    for(int i=q.front+1;i<=q.rear;i++)
    {
        printf("Voter id  : %d | Symbol : %c",q.v[i].id,q.v[i].symbol);
    }
    printf("\n");
}
void main()
{
    struct queue q;
    int choice;
    printf("Enter size of queue : ");
    scanf("%d",&q.size);
    q.front=-1;
    q.rear=-1;
    q.v=(struct vote *)malloc(q.size*sizeof(struct vote));
    do
    {
    printf("1.Enqueue\n");
    printf("2.Dequeue\n");
    printf("3.Display\n");
    printf("4.Exit\n");
    printf("Enter choice : ");
    scanf("%d",&choice);
        switch(choice)
        {
            case 1:enqueue(&q);
            break;
            case 2:dequeue(&q);
            break;
            case 3:display(q);
            break;
            case 4:printf("Exiting....\n");
            break;
        }
    } while (choice!=4);
```

```
}
```

```
Enter symbol : A
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice : 3
Voter id  : 1 | Symbol : AVoter id  : 2 | Symbol : BVoter id  : 3 | Symbol : A
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice : 2
Verification process
Voter id : 1
Symbol : A
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice : 2
Verification process
Voter id : 2
Symbol : B
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter choice : 2
Verification process
Voter id : 3
Symbol : A
1.Enqueue
2.Dequeue
```

```c
/*Develop a queue using linked lists to manage patients in a hospital
emergency room.
The system should prioritize patients based on the severity of their
condition and manage multiple doctors.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct queue {
    char name[20];
    int severity;
    struct queue *next;
} *front = NULL, *rear = NULL;
void enqueue() {
    struct queue *newPatient = (struct queue *)malloc(sizeof(struct
queue));
    if (newPatient == NULL) {
        printf("Queue is full\n");
        return;
    }
```

```
    printf("Enter name: ");
    getchar();
    fgets(newPatient->name, sizeof(newPatient->name), stdin);
    newPatient->name[strcspn(newPatient->name, "\n")] = '\0';

    printf("Enter severity (1 = Critical, 2 = Serious, 3 = Stable): ");
    scanf("%d", &newPatient->severity);
    newPatient->next = NULL;


    if (front == NULL || newPatient->severity < front->severity) {
        newPatient->next = front;
        front = newPatient;
        if (rear == NULL) rear = newPatient;
    }
    else
    {
        struct queue *t = front;
        while (t->next != NULL && t->next->severity <=
newPatient->severity) {
            t = t->next;
        }
        newPatient->next = t->next;
        t->next = newPatient;
        if (newPatient->next == NULL) rear = newPatient;
    }
}

void dequeue() {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    struct queue *t = front;
    char doc[20];
    printf("Enter doctor's name: ");
    getchar();
    fgets(doc, sizeof(doc), stdin);
    doc[strcspn(doc, "\n")] = '\0';
```

```
    printf("Doctor %s is treating %s with severity %d\n", doc, t->name,
t->severity);
    front = front->next;
    if (front == NULL) rear = NULL;
    free(t);
}

void display() {
    if (front == NULL) {
        printf("No patients in the waiting list.\n");
        return;
    }
    struct queue *t = front;
    printf("\nPatient Queue (By Priority):\n");
    while (t != NULL) {
        printf("Name: %s | Severity: %d\n", t->name, t->severity);
        t = t->next;
    }
}
int main() {
    int choice;
    do {
        printf("\n1. Add Patient (Enqueue)\n");
        printf("2. Process Patient (Dequeue)\n");
        printf("3. View Waiting List\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
```

```
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

```
1. Add Patient (Enqueue)
2. Process Patient (Dequeue)
3. View Waiting List
4. Exit
Enter choice: 3

Patient Queue (By Priority):
Name: blu | Severity: 1
Name: green | Severity: 2
Name: red | Severity: 3

1. Add Patient (Enqueue)
2. Process Patient (Dequeue)
3. View Waiting List
4. Exit
Enter choice: 2
Enter doctor's name: dre
Doctor dre is treating blu with severity 1

1. Add Patient (Enqueue)
2. Process Patient (Dequeue)
3. View Waiting List
4. Exit
Enter choice: 3

Patient Queue (By Priority):
Name: green | Severity: 2
Name: red | Severity: 3

1. Add Patient (Enqueue)
2. Process Patient (Dequeue)
3. View Waiting List
4. Exit
Enter choice:
```

```
/*Political Survey Data Collection**: Implement a queue using arrays to
manage data collection for a political survey.
```

```c
/*The system should handle multiple surveyors collecting data simultaneously
and ensure data consistency.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 100


struct SurveyResponse {
    char respondentName[50];
    int age;
    char politicalParty[50];
    char opinion[100];
};


struct Queue {
    struct SurveyResponse data[MAX];
    int front, rear;
};


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 == -1 || q->front > q->rear);
}


void enqueue(struct Queue *q) {
```

```c
    if (isFull(q)) {
        printf("Survey data queue is full! Cannot add more responses.\n");
        return;
    }

    struct SurveyResponse newResponse;
    printf("Enter respondent name: ");
    getchar();
    fgets(newResponse.respondentName, sizeof(newResponse.respondentName),
stdin);
    newResponse.respondentName[strcspn(newResponse.respondentName, "\n")]
= '\0';

    printf("Enter respondent age: ");
    scanf("%d", &newResponse.age);
    getchar(); // Consume newline

    printf("Enter political party affiliation: ");
    fgets(newResponse.politicalParty, sizeof(newResponse.politicalParty),
stdin);
    newResponse.politicalParty[strcspn(newResponse.politicalParty, "\n")]
= '\0';

    printf("Enter opinion on political issues: ");
    fgets(newResponse.opinion, sizeof(newResponse.opinion), stdin);
    newResponse.opinion[strcspn(newResponse.opinion, "\n")] = '\0';

    if (q->front == -1) q->front = 0;
    q->rear++;
    q->data[q->rear] = newResponse;

    printf("Survey response recorded successfully!\n");
}


void dequeue(struct Queue *q) {
    if (isEmpty(q)) {
        printf("No survey responses to process.\n");
        return;
    }
```

```
    struct SurveyResponse processedResponse = q->data[q->front];
    printf("\nProcessing Survey Response:\n");
    printf("Name: %s\n", processedResponse.respondentName);
    printf("Age: %d\n", processedResponse.age);
    printf("Political Party: %s\n", processedResponse.politicalParty);
    printf("Opinion: %s\n", processedResponse.opinion);

    q->front++;
    if (q->front > q->rear) {
        q->front = -1;
        q->rear = -1;
    }
}

void display(struct Queue *q) {
    if (isEmpty(q)) {
        printf("No survey responses collected yet.\n");
        return;
    }

    printf("\nSurvey Responses:\n");
    for (int i = q->front; i <= q->rear; i++) {
        printf("Respondent: %s, Age: %d, Party: %s, Opinion: %s\n",
               q->data[i].respondentName, q->data[i].age,
q->data[i].politicalParty, q->data[i].opinion);
    }
}


int main() {
    struct Queue surveyQueue;
    initQueue(&surveyQueue);
    int choice;

    do {
        printf("\nPolitical Survey Data Collection\n");
        printf("1. Collect Survey Response (Enqueue)\n");
        printf("2. Process Survey Response (Dequeue)\n");
        printf("3. View Collected Data\n");
```

```
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                enqueue(&surveyQueue);
                break;
            case 2:
                dequeue(&surveyQueue);
                break;
            case 3:
                display(&surveyQueue);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

```
espondent: blu, Age: 20, Party: B, Opinion: am,

olitical Survey Data Collection
. Collect Survey Response (Enqueue)
. Process Survey Response (Dequeue)
. View Collected Data
. Exit
nter choice: 2

rocessing Survey Response:
ame: red
ge: 19
olitical Party: A
pinion: adgsaf

olitical Survey Data Collection
. Collect Survey Response (Enqueue)
. Process Survey Response (Dequeue)
. View Collected Data
. Exit
nter choice: 2

rocessing Survey Response:
ame: blu
ge: 20
olitical Party: B
pinion: am,bfmqg,

olitical Survey Data Collection
. Collect Survey Response (Enqueue)
. Process Survey Response (Dequeue)
. View Collected Data
. Exit
nter choice:
```

```
/*Use linked lists to implement a queue for analyzing financial market
data.
The system should handle large volumes of data, perform real-time
analysis, and generate insights for decision-making.*/
```

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


struct MarketData {
    char symbol[10];
    float price;
    char timestamp[20];
    struct MarketData *next;
};


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


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


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


void enqueue(struct Queue *q) {
    struct MarketData *newData = (struct MarketData *)malloc(sizeof(struct
MarketData));

    if (newData == NULL) {
        printf("Memory allocation failed! Unable to add new data.\n");
        return;
    }

    printf("Enter stock symbol (e.g., AAPL, TSLA): ");
    scanf("%s", newData->symbol);
```

```
    printf("Enter stock price: ");
    scanf("%f", &newData->price);

    printf("Enter timestamp (HH:MM:SS): ");
    scanf("%s", newData->timestamp);

    newData->next = NULL;

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

    printf("Market data recorded successfully!\n");
}


void dequeue(struct Queue *q) {
    if (isEmpty(q)) {
        printf("No market data available for processing.\n");
        return;
    }

    struct MarketData *temp = q->front;
    printf("\nProcessing Market Data:\n");
    printf("Symbol: %s | Price: $%.2f | Time: %s\n", temp->symbol,
temp->price, temp->timestamp);

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

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

    free(temp);
}


void display(struct Queue *q) {
```

```
    if (isEmpty(q)) {
        printf("No market data available.\n");
        return;
    }

    printf("\nMarket Data Queue:\n");
    struct MarketData *temp = q->front;
    while (temp != NULL) {
        printf("Symbol: %s | Price: $%.2f | Time: %s\n", temp->symbol,
temp->price, temp->timestamp);
        temp = temp->next;
    }
}


void calculateMovingAverage(struct Queue *q) {
    if (isEmpty(q)) {
        printf("No data available for moving average calculation.\n");
        return;
    }

    struct MarketData *temp = q->front;
    int count = 0;
    float sum = 0;

    while (temp != NULL) {
        sum += temp->price;
        count++;
        temp = temp->next;
    }

    if (count > 0)
        printf("Moving Average of Stock Prices: $%.2f\n", sum / count);
    else
        printf("Not enough data for moving average calculation.\n");
}


int main() {
    struct Queue marketQueue;
```

```c
    initQueue(&marketQueue);
    int choice;

    do {
        printf("\nFinancial Market Data Processing\n");
        printf("1. Add Market Data (Enqueue)\n");
        printf("2. Process Oldest Data (Dequeue)\n");
        printf("3. View Market Data Queue\n");
        printf("4. Calculate Moving Average of Prices\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                enqueue(&marketQueue);
                break;
            case 2:
                dequeue(&marketQueue);
                break;
            case 3:
                display(&marketQueue);
                break;
            case 4:
                calculateMovingAverage(&marketQueue);
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

```
4. Calculate Moving Average of Prices
5. Exit
Enter choice: 4
Moving Average of Stock Prices: $350.00

Financial Market Data Processing
1. Add Market Data (Enqueue)
2. Process Oldest Data (Dequeue)
3. View Market Data Queue
4. Calculate Moving Average of Prices
5. Exit
Enter choice: 2

Processing Market Data:
Symbol: TSLA | Price: $200.00 | Time: 10:38:22

Financial Market Data Processing
1. Add Market Data (Enqueue)
2. Process Oldest Data (Dequeue)
3. View Market Data Queue
4. Calculate Moving Average of Prices
5. Exit
Enter choice: 3

Market Data Queue:
Symbol: AAPL | Price: $500.00 | Time: 07:12:45

Financial Market Data Processing
1. Add Market Data (Enqueue)
2. Process Oldest Data (Dequeue)
3. View Market Data Queue
4. Calculate Moving Average of Prices
5. Exit
Enter choice:
```