```cpp
/**
 * Lab Sheet - 3, Queue modeling of Nepali Cinema Ticket Booking System.
 *
 * By Santa Basnet.
 * Everest Engineering College.
 * Date: 2021-11-28
 */

#include <cstdio>
#include <cstring>

#define MAX_FAMILY_SEATS 100
#define MAX_SINGLE_SEATS 100

const int TRUE = 1;
const int FALSE = 0;

/**
 * Ticket type identity.
 */
enum TicketType {
    FAMILY = 1, SINGLE = 2
};

/**
 * Ticket representation.
 */
struct Ticket {
    char phoneNumber[15];
    int number;
    int type;
    long arrivalTime;
};

/**
 * Reservation type Tickets Queue for Reservation.
 */
struct Reservation {
    struct Ticket tickets[MAX_FAMILY_SEATS];
    int front;
    int rear;
};

/**
 * Structure that represents Cinema Tickets of family and single type
 * reservations.
 */
struct Cinema {
    struct Reservation familyReservation;
    struct Reservation singleReservation;
    int singleReservedCount;
    int familyReservedCount;
} nepaliCinema;

/**
 * Returns total capacity of all ticket types.
 */
int totalCapacity() {
```

```
    return MAX_FAMILY_SEATS + MAX_SINGLE_SEATS;
}

/**
 * Allocates single type ticket number.
 * @return ticketNumber.
 */
int allocateSingleTicketNumber() {
    return ++nepaliCinema.singleReservedCount;
}

/**
 * Allocates single type ticket number.
 * @return ticketNumber.
 */
int allocateFamilyTicketNumber() {
    return ++nepaliCinema.familyReservedCount;
}

/**
 * Returns the ticket literals for the given index.
 * @param ticketType
 * @return ticketLiteral
 */
const char *ticketTypeOf(int ticketType) {
    switch (ticketType) {
        case FAMILY:
            return "FAMILY";
        case SINGLE:
            return "SINGLE";
    }
}

/*
 * Print Ticket Information.
 */
void displayTicket(struct Ticket ticket) {
    printf(
            "\n%4d | %20s | %16s | %4ld Unit",
            ticket.number, ticket.phoneNumber, ticketTypeOf(ticket.type),
ticket.arrivalTime
    );
}

/**
 * Checks if the family queue entry is a first enqueue operation.
 */
int isFirstFamilyReservation() {
    return nepaliCinema.familyReservation.front == -1 &&
nepaliCinema.familyReservation.rear == -1;
}

/**
 * Checks if the family queue is full or not.
 */
int isFamilyReservationFull() {
    return (nepaliCinema.familyReservation.rear + 1) % MAX_FAMILY_SEATS ==
nepaliCinema.familyReservation.front;
```

```
}

/**
 * Checks if the single queue entry is a first enqueue operation.
 */
int isFirstSingleReservation() {
    return nepaliCinema.singleReservation.front == -1 &&
nepaliCinema.singleReservation.rear == -1;
}

/**
 * Checks if the family queue is full or not.
 */
int isSingleReservationFull() {
    return (nepaliCinema.singleReservation.rear + 1) % MAX_SINGLE_SEATS ==
nepaliCinema.singleReservation.front;
}

/**
 * Checks if the ticket is already booked or not for the given phone number.
 * @param ticket
 * @return 1 for booked, 0 otherwise.
 */
int isAlreadyBooked(struct Ticket ticket) {
    /**
     * Search in family reservation.
     */
    int index = nepaliCinema.familyReservation.front;
    while (index <= nepaliCinema.familyReservation.rear) {
        int isEqual = strcmp(nepaliCinema.familyReservation.tickets[index].phoneNumber,
ticket.phoneNumber) == 0;
        if (isEqual) return TRUE;
        index = (index + 1) % MAX_FAMILY_SEATS;
    }

    /**
     * Search in single reservation.
     */
    index = nepaliCinema.singleReservation.front;
    while (index <= nepaliCinema.singleReservation.rear) {
        int isEqual = strcmp(nepaliCinema.singleReservation.tickets[index].phoneNumber,
ticket.phoneNumber) == 0;
        if (isEqual) return TRUE;
        index = (index + 1) % MAX_SINGLE_SEATS;
    }
    /**
     * Return not found result.
     */
    return FALSE;
}

/**
 * Perform enqueue operation of family type reservation.
 * @param ticket
 */
int enqueueFamily(struct Ticket ticket) {
    if (isFirstFamilyReservation()) {
        nepaliCinema.familyReservation.front = 0;
        nepaliCinema.familyReservation.rear = 0;
```

```
        nepaliCinema.familyReservation.tickets[nepaliCinema.familyReservation.rear] =
ticket;
        return TRUE;
    } else if (isAlreadyBooked(ticket)) {
        printf("\n%s has already booked the ticket.", ticket.phoneNumber);
        return FALSE;
    } else if (isFamilyReservationFull()) {
        printf("\nAll family seats are booked.");
        return FALSE;
    } else {
        nepaliCinema.familyReservation.rear = (nepaliCinema.familyReservation.rear + 1) %
MAX_FAMILY_SEATS;
        nepaliCinema.familyReservation.tickets[nepaliCinema.familyReservation.rear] =
ticket;
        return TRUE;
    }
}

/**
 * Perform enqueue operation of single type reservation.
 * @param ticket
 */
int enqueueSingle(struct Ticket ticket) {
    if (isFirstSingleReservation()) {
        nepaliCinema.singleReservation.front = 0;
        nepaliCinema.singleReservation.rear = 0;
        nepaliCinema.singleReservation.tickets[nepaliCinema.singleReservation.rear] =
ticket;
        return TRUE;
    } else if (isAlreadyBooked(ticket)) {
        printf("\n%s has already booked the ticket.", ticket.phoneNumber);
        return FALSE;
    } else if (isSingleReservationFull()) {
        printf("\nAll single seats are booked.");
        return FALSE;
    } else {
        nepaliCinema.singleReservation.rear = (nepaliCinema.singleReservation.rear + 1) %
MAX_SINGLE_SEATS;
        nepaliCinema.singleReservation.tickets[nepaliCinema.singleReservation.rear] =
ticket;
        return TRUE;
    }
}

/**
 * Perform enqueue operation, for generalization.
 * @param ticket
 */
void enQueue(struct Ticket ticket) {
    switch (ticket.type) {
        case FAMILY: {
            enqueueFamily(ticket);
            break;
        }
        case SINGLE: {
            enqueueSingle(ticket);
            break;
        }
    }
```

```
}

/**
 * Verifies if the family queue has only one element.
 * @return 1 for one reservation, 0 otherwise.
 */
int hasOneFamilyReservation() {
    return nepaliCinema.familyReservation.front == nepaliCinema.familyReservation.rear;
}

/**
 * Verifies if the single queue has only one element.
 * @return 1 for one reservation, 0 otherwise.
 */
int hasOneSingleReservation() {
    return nepaliCinema.singleReservation.front == nepaliCinema.singleReservation.rear;
}

/**
 * Perform dequeue operation on family reservation.
 * @return 1 for successful dequeue operation, 0 otherwise.
 */
int dequeueFamily() {
    if (isFirstFamilyReservation()) {
        printf("\nNo family reservation tickets are booked till now.");
        return FALSE;
    } else if (hasOneFamilyReservation()) {
        nepaliCinema.familyReservation.front = -1;
        nepaliCinema.familyReservation.rear = -1;
        nepaliCinema.familyReservedCount = 0;
        return TRUE;
    } else {
        int index = (nepaliCinema.familyReservation.front + 1) % MAX_FAMILY_SEATS;
        nepaliCinema.familyReservation.front = index;
        nepaliCinema.familyReservedCount--;
        return TRUE;
    }
}

/**
 * Perform dequeue operation on single reservation.
 * @return 1 for successful dequeue operation.
 */
int dequeueSingle() {
    if (isFirstSingleReservation()) {
        printf("\nNo single reservation tickets are booked till now.");
        return FALSE;
    } else if (hasOneSingleReservation()) {
        nepaliCinema.singleReservation.front = -1;
        nepaliCinema.singleReservation.rear = -1;
        nepaliCinema.singleReservedCount = 0;
        return TRUE;
    } else {
        int index = (nepaliCinema.singleReservation.front + 1) % MAX_SINGLE_SEATS;
        nepaliCinema.singleReservation.front = index;
        nepaliCinema.singleReservedCount--;
        return TRUE;
    }
}
```

```
/**
 * Perform dequeue operation, for generalization.
 * @param ticket
 */
void deQueue(int ticketType) {
    switch (ticketType) {
        case FAMILY: {
            dequeueFamily();
            break;
        }
        case SINGLE: {
            dequeueSingle();
            break;
        }
    }
}

/**
 * Display Ticket Queues.
 */
void displayFamilyQueue() {
    if (isFirstFamilyReservation()) {
        printf("\nEmpty family reservations.");
    } else {
        printf("\n\nFamily Reservations: ");
        int index = nepaliCinema.familyReservation.front;
        while (index <= nepaliCinema.familyReservation.rear) {
            displayTicket(nepaliCinema.familyReservation.tickets[index]);
            index = (index + 1) % MAX_FAMILY_SEATS;
        }
    }
}

/**
 * Display Ticket Queues.
 */
void displaySingleQueue() {
    if (isFirstSingleReservation()) {
        printf("\nEmpty single reservations.");
    } else {
        printf("\n\nSingle Reservations:");
        int index = nepaliCinema.singleReservation.front;
        while (index <= nepaliCinema.singleReservation.rear) {
            displayTicket(nepaliCinema.singleReservation.tickets[index]);
            index = (index + 1) % MAX_SINGLE_SEATS;
        }
    }
}

/**
 * Initialize all the tickets.
 */
void initializeNewShow() {
    /**
     * Initialize Reserved Numbers.
     */
    nepaliCinema.familyReservedCount = 0;
    nepaliCinema.singleReservedCount = 0;
```

```
    /**
     * Initialize family reservation.
     */
    nepaliCinema.familyReservation.front = -1;
    nepaliCinema.familyReservation.rear = -1;

    /**
     * Initialize single reservation.
     */
    nepaliCinema.singleReservation.front = -1;
    nepaliCinema.singleReservation.rear = -1;

    struct Ticket ticketPerson1 = {
            "+9779849023236",
            allocateSingleTicketNumber(),
            SINGLE,
            10
    };

    struct Ticket ticketPerson2 = {
            "+9779849023260",
            allocateFamilyTicketNumber(),
            FAMILY,
            21
    };

    struct Ticket ticketPerson3 = {
            "+9779849023299",
            allocateFamilyTicketNumber(),
            FAMILY,
            22
    };

    struct Ticket ticketPerson4 = {
            "+9779849023220",
            allocateSingleTicketNumber(),
            SINGLE,
            24
    };

    struct Ticket ticketPerson5 = {
            "+9779849023243",
            allocateFamilyTicketNumber(),
            FAMILY,
            27
    };

    enQueue(ticketPerson1);
    enQueue(ticketPerson2);
    enQueue(ticketPerson3);
    enQueue(ticketPerson4);
    enQueue(ticketPerson5);

    //Double booked.
    enQueue(ticketPerson3);
}
```

```c
/**
 * Returns single reservation.
 */
int totalSingleReservation() {
    return nepaliCinema.singleReservedCount;
}

/**
 * Returns family reservation.
 */
int totalFamilyReservation() {
    return nepaliCinema.familyReservedCount;
}

/**
 * Returns total allocated seats.
 */
int totalAllocatedSeats() {
    return totalSingleReservation() + totalFamilyReservation();
}

/**
 * Returns total available Seats.
 */
int aggregatedAvailableSeats() {
    return totalCapacity() - totalAllocatedSeats();
}

/**
 * Displays the total available seats.
 */
void displayAvailableSeats() {
    printf("\n\nAvailable Seats: %4d.", aggregatedAvailableSeats());
}

/**
 * Main starts here and successfully exists with return 1.
 * @return 1
 */
int main() {
    initializeNewShow();
    displayFamilyQueue();
    displaySingleQueue();
    displayAvailableSeats();
    return 1;
}
```

**Output:**

```
+9779849023299 has already booked the ticket.

Family Reservations:
   1 |       +9779849023260 |           FAMILY |   21 Unit
   2 |       +9779849023299 |           FAMILY |   22 Unit
```

```
   3 |       +9779849023243 |           FAMILY |   27 Unit

Single Reservations:
   1 |       +9779849023236 |           SINGLE |   10 Unit
   2 |       +9779849023220 |           SINGLE |   24 Unit

Available Seats:  195.
```

***