```
//1.

# include <stdio.h>

void MyFun(){
   static int staticVar=0;
   const int  consvar=2;
   staticVar++;

   switch(consvar){
      case 1:
         printf("constVar  is 1\n");
         break;
      case 2:
         printf("constVar  is 2\n");
         break;
      case 3:
         printf("constVar  is 3\n");
         break;
      default:
         printf("constVar  is unknown\n");
   }

   printf("Static variable vaue: %d\n", staticVar);
}

int main(){
   MyFun();
   MyFun();
   MyFun();
}
```

//2.

```
# include <stdio.h>

void MyFun(){
   const int MAX_CALLS=4;
   static int callcount=0;
   callcount++;

   if(callcount>MAX_CALLS)
   {
      printf("Maximum  Limit Reached\n");
      return;
   }
   switch(callcount){
      case 1:
      printf("It  is the First Call\n");
      break;
      case 2:
      printf("It  is the second  Call\n");
      break;
      case 3:
      printf("Its  is the third Call\n");
      break;
      default:
      printf("This  is a call number  %d\n",callcount);
      break;
   }

}
```

```
int main(){
   MyFun();
   MyFun();
   MyFun();
   MyFun();
   MyFun();
}

//3.

include <stdio.h>

void arrayoperations() {
   const int ARRAY_SIZE = 5;
   static int arr[] = {2, 4, 6, 8, 9};
   int operation;

   printf("Choose an operation:\n");
   printf("1. Add\n");
   printf("2. Subtract\n");
   printf("3. Multiply\n");
   printf("4. Exit\n");
   printf("Enter the Operation (1-4): ");
   scanf("%d", &operation);

   if (operation < 1 || operation > 4) {
      printf("Invalid operation\n");
      return;
   }

   if (operation == 4) {
```

```
        printf("Exiting...\n");
        return;
    }

    switch (operation)  {
        case 1:
            for (int i = 0; i < ARRAY_SIZE; i++) {
                arr[i] += 1;
            }
            printf("Added  1 to each value\n");
            break;
        case 2:
            for (int i = 0; i < ARRAY_SIZE; i++) {
                arr[i] -= 1;
            }
            printf("Subtracted  1 from each value\n");
            break;
        case 3:
            for (int i = 0; i < ARRAY_SIZE; i++) {
                arr[i] *= 2;
            }
            printf("Multiplied  each value by 2\n");
            break;
    }

    printf("Updated  array: ");
    for (int i = 0; i < ARRAY_SIZE; i++) {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}
```

```
int main() {
    for (int i = 0; i < 5; i++) {
        arrayoperations();
    }
    return 0;
}

//4.
#include <stdio.h>

void checkThreshold();

int main() {
    for (int i = 0; i < 12; i++) {
        checkThreshold();
    }
    return 0;
}

void checkThreshold()  {
    static int counter = 0;
    const int THRESHOLD = 5;
    counter++;

    printf("\nExecution  count: %d\n", counter);

    switch (counter)  {
        case 1:
            printf("First execution of the switch case.\n");
            break;
```

```
        case 2:
            printf("Second execution of the switch case.\n");
            break;
        case 3:
            printf("Third execution of the switch case.\n");
            break;
        case 4:
            printf("Fourth execution of the switch case.\n");
            break;
        case 5:
            printf("Threshold reached! Counter is now %d.\n", counter);
            break;
        default:
            if (counter > THRESHOLD) {
                printf("Counter has exceeded the threshold! (Threshold: %d)\n", THRESHOLD);
            } else {
                printf("Other case.\n");
            }
            break;
    }

    if (counter >= THRESHOLD) {
        counter = 0;
        printf("Counter has been reset.\n");
    }
}

//5.
#include <stdio.h>
#define LIMIT 5
void incrementCounter() {
```

```
    static int counter = 0;
    counter++;

    switch (counter) {
        case 1:
            printf("Counter is at 1\n");
            break;
        case 2:
            printf("Counter is at 2\n");
            break;
        case 3:
            printf("Counter is at 3\n");
            break;
        case 4:
            printf("Counter is at 4\n");
            break;
        case 5:
            printf("Counter is at 5. Resetting...\n");
            counter = 0;
            break;
        default:
            printf("Counter reset to 0\n");
            break;
    }
}
int main() {
    for (int i = 0; i < 10; i++) {
        incrementCounter();
    }
    return 0;
}
```

```
//1.

#include <stdio.h>

void initializeArray(int  *arr, int size);
void modifyArray(int  *arr, int size);
void printArray(const  int *arr, int size);

int main() {
    const int SIZE = 5;
    int arr[SIZE];

    initializeArray(arr,  SIZE);
    printf("Array  before modification:\n");
    printArray(arr,  SIZE);

    modifyArray(arr,  SIZE);
    printf("Array  after modification:\n");
    printArray(arr,  SIZE);

    return 0;
}

void initializeArray(int  *arr, int size) {
    for (int i = 0; i < size; i++) {
        arr[i] = i + 1;
    }
}

void modifyArray(int  *arr, int size) {
```

```
    for (int i = 0; i < size; i++) {
        arr[i] *= 2;
    }
}

void printArray(const  int *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}

//2.
#include <stdio.h>

void transposeMatrix(int  matrix[3][3], int rows, int cols);
void printMatrix(const  int matrix[3][3], int rows, int cols);

int main() {
    const int rows = 3, cols = 3;
    int matrix[3][3], i, j;

    printf("Enter elements of 3x3 matrix:\n");
    for(i = 0; i < rows; i++) {
        for(j = 0; j < cols; j++) {
            scanf("%d",  &matrix[i][j]);
        }
    }

    printf("Original  Matrix:\n");
    printMatrix(matrix,  rows, cols);
```

```
    transposeMatrix(matrix,  rows, cols);

    printf("Transposed  Matrix:\n");
    printMatrix(matrix,  rows, cols);

    return 0;
}

void transposeMatrix(int  matrix[3][3], int rows, int cols) {
    int temp;
    for (int i = 0; i < rows; i++) {
        for (int j = i + 1; j < cols; j++) {
            temp = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = temp;
        }
    }
}

void printMatrix(const  int matrix[3][3], int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d  ", matrix[i][j]);
        }
        printf("\n");
    }
}

//3.
```

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

void initializeArray(int *arr, int size);
void printArray(const int *arr, int size);

int main() {
    const int SIZE = 10;
    int *arr = (int *)malloc(SIZE * sizeof(int));

    if (arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    initializeArray(arr, SIZE);
    printf("Array elements:\n");
    printArray(arr, SIZE);

    free(arr);
    return 0;
}

void initializeArray(int *arr, int size) {
    int i = 0;
    while (i < size) {
        arr[i] = i + 1;
        i++;
    }
}
```

```
void printArray(const  int *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}

//4.

#include <stdio.h>

void swapArrays(int  *arr1, int *arr2, int size);
void printArray(const  int *arr, int size);

int main() {
    const int SIZE = 5;
    int arr1[] = {11, 12, 13, 14, 15};
    int arr2[] = {6, 7, 8, 9, 10};

    printf("Arrays  before swapping:\n");
    printArray(arr1,  SIZE);
    printArray(arr2,  SIZE);

    swapArrays(arr1,  arr2, SIZE);

    printf("Arrays  after swapping:\n");
    printArray(arr1,  SIZE);
    printArray(arr2,  SIZE);

    return 0;
}
```

```
void swapArrays(int *arr1, int *arr2, int size) {
    int temp;
    for (int i = 0; i < size; i++) {
        temp = arr1[i];
        arr1[i] = arr2[i];
        arr2[i] = temp;
    }
}

void printArray(const int *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

//5.

#include <stdio.h>

void readString(char *str);
void countFrequency(const char *str);

int main() {
    char str[100];
    readString(str);
    countFrequency(str);
    return 0;
}
```

```
void readString(char *str) {
    printf("Enter a string: ");
    fgets(str, 100, stdin);
}

void countFrequency(const char *str) {
    int freq[256] = {0};
    int i = 0;

    do {
        freq[(int)str[i]]++;
        i++;
    } while (str[i] != '\0');

    printf("Character frequencies:\n");
    for (int i = 0; i < 256; i++) {
        if (freq[i] > 0) {
            printf("%c: %d\n", i, freq[i]);
        }
    }
}

//1.

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

typedef struct {
    char street[100];
    char city[100];
    int zip;
```

```
} Address;

typedef struct {
    int id;
    char name[100];
    Address address;  // Nested structure for Address
} Employee;

void inputEmployeeData(Employee *emp) {
    printf("Enter Employee ID: ");
    scanf("%d", &emp->id);
    getchar();

    printf("Enter Employee Name: ");
    scanf("%[^\n]s", emp->name);

    printf("Enter Street: ");
    getchar();
    scanf("%[^\n]s", emp->address.street);

    printf("Enter City: ");
    getchar();
    scanf("%[^\n]s", emp->address.city);


    printf("Enter Zip Code: ");
    scanf("%d", &emp->address.zip);
}

void displayEmployeeData(Employee emp) {
    printf("\nEmployee ID: %d\n", emp.id);
```

```
    printf("Employee  Name: %s\n", emp.name);
    printf("Address:  %s, %s, %d\n", emp.address.street,  emp.address.city, emp.address.zip);
}

int main() {
    const int SIZE = 2;
    Employee employees[SIZE];

    for (int i = 0; i < SIZE; i++) {
        printf("\nEnter  details for Employee  %d\n", i + 1);
        inputEmployeeData(&employees[i]);
    }

    for (int i = 0; i < SIZE; i++) {
        printf("\nDisplaying  details of Employee %d:\n", i + 1);
        displayEmployeeData(employees[i]);
    }

    return 0;
}

//2.

#include <stdio.h>

typedef union {
    int i;
    float f;
    char str[100];
} Data;
```

```
typedef struct {
    char name[100];
    Data data;  // Nested union to store data
} Information;

void inputInformation(Information *info) {
    printf("Enter name: ");
    scanf("%[^\n]s", info->name);

    printf("Enter type of data (1 for i, 2 for f, 3 for str): ");
    int choice;
    scanf("%d", &choice);
    getchar();

    if (choice == 1) {
        printf("Enter an integer: ");
        scanf("%d", &info->data.i);
    } else if (choice == 2) {
        printf("Enter a float: ");
        scanf("%f", &info->data.f);
    } else if (choice == 3) {
        printf("Enter a string: ");
        scanf("%[^\n]s", info->data.str);
    }
}

void displayInformation(Information info) {
    printf("Name: %s\n", info.name);
    printf("Data: ");
    if (info.data.i != 0) {
        printf("%d (int)\n", info.data.i);
```

```
    } else if (info.data.f != 0.0) {
        printf("%f (float)\n", info.data.f);
    } else {
        printf("%s (string)\n", info.data.str);
    }
}

int main() {
    Information info;
    inputInformation(&info);
    displayInformation(info);
    return 0;
}

//3.

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

typedef union {
    int length;
    char reversed[100];
} StringData;

typedef struct {
    char name[100];
    StringData data; // Nested union to store either length or reversed string
} StringInfo;

void inputStringInfo(StringInfo *info) {
    printf("Enter a string: ");
```

```
    scanf("%[^\n]s", info->name);

    printf("Enter choice (1 for length, 2 for reversed): ");
    int choice;
    scanf("%d", &choice);

    if (choice == 1) {
        info->data.length = strlen(info->name);
    } else if (choice == 2) {
        int len = strlen(info->name);
        for (int i = 0; i < len; i++) {
            info->data.reversed[i] = info->name[len - 1 - i];
        }
        info->data.reversed[len] = '\0';
    }
}

void displayStringInfo(StringInfo info) {
    printf("String: %s\n", info.name);
    printf("Choice result: ");
    if (info.data.length > 0) {
        printf("Length: %d\n", info.data.length);
    } else {
        printf("Reversed: %s\n", info.data.reversed);
    }
}

int main() {
    StringInfo info;
    inputStringInfo(&info);
    displayStringInfo(info);
```

```
    return 0;
}

//4.

#include <stdio.h>

typedef union {
    int i;
    float f;
} NestedUnion;

typedef struct {
    int id;
    NestedUnion data;  // Nested union within structure
} ComplexStructure;

void inputComplexStructure(ComplexStructure  *cs) {
    printf("Enter ID: ");
    scanf("%d", &cs->id);
    printf("Enter data type (1 for int, 2 for float): ");
    int choice;
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter an integer: ");
        scanf("%d", &cs->data.i);
    } else {
        printf("Enter a float: ");
        scanf("%f", &cs->data.f);
    }
```

```
}

void displayComplexStructure(ComplexStructure cs) {
    printf("ID: %d\n", cs.id);
    printf("Data: ");
    if (cs.data.i != 0) {
        printf("%d (int)\n", cs.data.i);
    } else {
        printf("%f (float)\n", cs.data.f);
    }
}

int main() {
    ComplexStructure cs;
    inputComplexStructure(&cs);
    displayComplexStructure(cs);
    return 0;
}

//5.

#include <stdio.h>

typedef union {
    int pages;
    int year;
} BookUnion;

typedef struct {
    char name[100];
    char author[100];
```

```
    BookUnion bookInfo;
} Book;

void inputBookData(Book *book) {
    printf("Enter book name: ");
    scanf("%[^\n]s", book->name);

    printf("Enter author name: ");
    scanf("%[^\n]s", book->author);

    printf("Enter choice (1 for pages, 2 for year): ");
    int choice;
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter number of pages: ");
        scanf("%d", &book->bookInfo.pages);
    } else {
        printf("Enter publication year: ");
        scanf("%d", &book->bookInfo.year);
    }
}

void displayBookData(Book book) {
    printf("\nBook Name: %s\n", book.name);
    printf("Author: %s\n", book.author);
    if (book.bookInfo.pages != 0) {
        printf("Pages: %d\n", book.bookInfo.pages);
    } else {
        printf("Publication Year: %d\n", book.bookInfo.year);
    }
```

```
}

int main() {
    Book book;
    inputBookData(&book);
    displayBookData(book);
    return 0;
}

//1.

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

#define MAX 5

typedef struct {
    int arr[MAX];
    int top;
} Stack;

void initStack(Stack* stack) {
    stack->top = -1;
}

int isFull(Stack* stack) {
    return stack->top == MAX - 1;
}

int isEmpty(Stack* stack) {
    return stack->top == -1;
```

```
}

void push(Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack Overflow\n");
        return;
    }
    stack->arr[++(stack->top)] = value;
}

int pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack->arr[(stack->top)--];
}

int peek(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is Empty\n");
        return -1;
    }
    return stack->arr[stack->top];
}

int main() {
    Stack stack;
    initStack(&stack);

    push(&stack, 10);
```

```
    push(&stack, 20);
    push(&stack, 30);

    printf("Top element is %d\n", peek(&stack));

    printf("Popped element is %d\n", pop(&stack));
    printf("Top element after pop is %d\n", peek(&stack));

    return 0;
}

//2.

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

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

Node* top = NULL;

int isEmpty() {
    return top == NULL;
}

void push(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory Allocation Error\n");
```

```
        return;
    }
    newNode->data = value;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (isEmpty()) {
        printf("Stack Underflow\n");
        return -1;
    }
    Node* temp = top;
    int poppedValue = top->data;
    top = top->next;
    free(temp);
    return poppedValue;
}

int peek() {
    if (isEmpty()) {
        printf("Stack is Empty\n");
        return -1;
    }
    return top->data;
}

int main() {
    push(10);
    push(20);
    push(30);
```

```
    printf("Top element is %d\n", peek());

    printf("Popped element is %d\n", pop());
    printf("Top element after pop is %d\n", peek());

    return 0;
}

//3.

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

#define MAX 5

typedef struct {
    int arr[MAX];
    int top;
} Stack;

void initStack(Stack* stack) {
    stack->top = -1;
}

int isFull(Stack* stack) {
    return stack->top == MAX - 1;
}

int isEmpty(Stack* stack) {
    return stack->top == -1;
```

```
}

void push(Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack Overflow\n");
        return;
    }
    stack->arr[++(stack->top)] = value;
}

int pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack->arr[(stack->top)--];
}

void reverseStack(Stack* stack) {
    if (isEmpty(stack)) return;

    int topElement = pop(stack);
    reverseStack(stack);
    push(stack, topElement);
}

void printStack(Stack* stack) {
    for (int i = 0; i <= stack->top; i++) {
        printf("%d ", stack->arr[i]);
    }
    printf("\n");
```

```
}

int main() {
    Stack stack;
    initStack(&stack);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);

    printf("Stack before reversal: ");
    printStack(&stack);

    reverseStack(&stack);

    printf("Stack after reversal: ");
    printStack(&stack);

    return 0;
}

//4.

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

typedef struct Node {
    int data;
    struct Node* next;
} Node;
```

```
Node* top1 = NULL;
Node* top2 = NULL;

int isEmpty(Node* top) {
    return top == NULL;
}

void push(Node** top, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = *top;
    *top = newNode;
}

int pop(Node** top) {
    if (isEmpty(*top)) {
        printf("Stack Underflow\n");
        return -1;
    }
    Node* temp = *top;
    int poppedValue = (*top)->data;
    *top = (*top)->next;
    free(temp);
    return poppedValue;
}

void mergeStacks(Node** top1, Node** top2) {
    while (*top2 != NULL) {
        push(top1, pop(top2));
    }
}
```

```
void printStack(Node* top) {
    while (top != NULL) {
        printf("%d ", top->data);
        top = top->next;
    }
    printf("\n");
}

int main() {
    push(&top1, 10);
    push(&top1, 20);
    push(&top1, 30);

    push(&top2, 40);
    push(&top2, 50);
    push(&top2, 60);

    printf("Stack 1 before merging: ");
    printStack(top1);
    printf("Stack 2 before merging: ");
    printStack(top2);

    mergeStacks(&top1, &top2);

    printf("Stack 1 after merging: ");
    printStack(top1);

    return 0;
}
```

```
//5.

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

#define MAX 100

typedef struct {
    char arr[MAX];
    int top;
} Stack;

void initStack(Stack* stack) {
    stack->top = -1;
}

int isFull(Stack* stack) {
    return stack->top == MAX - 1;
}

int isEmpty(Stack* stack) {
    return stack->top == -1;
}

void push(Stack* stack, char value) {
    if (isFull(stack)) {
        printf("Stack Overflow\n");
        return;
    }
    stack->arr[++(stack->top)] = value;
}
```

```
char pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack->arr[(stack->top)--];
}

int isBalanced(char* expression) {
    Stack stack;
    initStack(&stack);
    char ch;

    for (int i = 0; expression[i] != '\0'; i++) {
        ch = expression[i];
        if (ch == '(' || ch == '{' || ch == '[') {
            push(&stack, ch);
        } else if (ch == ')' || ch == '}' || ch == ']') {
            if (isEmpty(&stack)) {
                return 0; // Unbalanced
            }
            char top = pop(&stack);
            if ((ch == ')' && top != '(') || (ch == '}' && top != '{') || (ch == ']' && top != '[')) {
                return 0; // Unbalanced
            }
        }
    }

    return isEmpty(&stack); // If stack is empty, balanced
}
```

```
int main() {
    char expression[100];
    printf("Enter an expression: ");
    scanf("%s", expression);

    if (isBalanced(expression)) {
        printf("Balanced\n");
    } else {
        printf("Unbalanced\n");
    }

    return 0;
}

//6.

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

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

Node* top = NULL;

int isEmpty() {
    return top == NULL;
}
```

```
void push(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (isEmpty()) {
        printf("Stack Underflow\n");
        return -1;
    }
    Node* temp = top;
    int poppedValue = top->data;
    top = top->next;
    free(temp);
    return poppedValue;
}

int evaluatePostfix(char* expression) {
    for (int i = 0; expression[i] != '\0'; i++) {
        if (isdigit(expression[i])) {
            push(expression[i] - '0');
        } else {
            int operand2 = pop();
            int operand1 = pop();
            switch (expression[i]) {
                case '+': push(operand1 + operand2); break;
                case '-': push(operand1 - operand2); break;
                case '*': push(operand1 * operand2); break;
```

```
            case '/': push(operand1 / operand2); break;
        }
      }
   }
   return pop();
}

int main() {
   char expression[100];
   printf("Enter a postfix expression: ");
   scanf("%s", expression);

   int result = evaluatePostfix(expression);
   printf("Result = %d\n", result);

   return 0;
}



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

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

void create(struct Queue *, int);
void enqueue (struct Queue *, int);
```

```
void display(struct  Queue);
int dequeue(struct  Queue *);


int main(){
   struct Queue q;
   create(&q,5);
   enqueue(&q,7);
   enqueue(&q,8);
   enqueue(&q,9);
   display(q);
   printf("%d ",dequeue(&q));
   printf("\n");
   display(q);
   return 0;
}

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

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

```
void display(struct  Queue q){
   int i;
   for(i = q.front+1;i<=q.rear;i++){
      printf("%d  -> ",q.Q[i]);
   }
   printf("\n");
}

int dequeue(struct  Queue *q){
   int x = -1;
   if(q->front == q->rear){
      printf("Queue  is Empty");
   }else{
      q->front++;
      x = q->Q[q->front];
   }
   return x;
}



//1.

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

struct Student{
   int size;
   int front;
   int rear;
   int *Q;
```

```
};

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



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

   enqueue(&q,100);
   enqueue(&q,101);
   enqueue(&q,102);
   display(q);

   printf("Admitted Student: %d\n", dequeue(&q));

   display(q);
   return 0;
}

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

void enqueue(struct Student *q,int student_id){
```

```
    if(q->rear==q->size-1){
        printf("Queue  is full\n");
    }
    else{
        q->rear++;
        q->Q[q->rear] = student_id;
    }
}

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

void display(struct  Student  q){
    if (q.front  == q.rear) {
        printf("Queue  is empty\n");
    } else {
        for (int i = q.front  + 1; i <= q.rear; i++) {
            printf("Student  ID: %d\n",  q.Q[i]);
        }
    }
}*/

//1.
```

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

struct Student{
   int size;
   int front;
   int rear;
   int *Q;
};

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



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

   enqueue(&q,100);
   enqueue(&q,101);
   enqueue(&q,102);
   display(q);

   printf("Admitted Student: %d\n", dequeue(&q));

   display(q);
   return 0;
```

```
}

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

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

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

void display(struct Student q){
   if (q.front == q.rear) {
```

```
        printf("Queue  is empty\n");
    } else {
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Student  ID: %d\n", q.Q[i]);
        }
    }
}

//2.
#include <stdio.h>
#include <stdlib.h>

struct LibraryQueue {
    int size;
    int front;
    int rear;
    int *Q;
};

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

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

    enqueue(&q, 1001);
    enqueue(&q, 1002);
    enqueue(&q, 1003);
```

```
    display(q);

    printf("Student with ID %d borrowed a book.\n", dequeue(&q));
    display(q);
    return 0;
}

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

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

int dequeue(struct LibraryQueue *q) {
    int x = -1;
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        x = q->Q[q->front];
    }
    return x;
```

```
}

void display(struct LibraryQueue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
    } else {
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Student ID: %d\n", q.Q[i]);
        }
    }
}

//3.

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

struct CafeteriaQueue {
    int size;
    int front;
    int rear;
    int *Q;
};

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

int main() {
    struct CafeteriaQueue q;
```

```
    create(&q, 5);

    enqueue(&q, 1001);
    enqueue(&q, 1002);
    enqueue(&q, 1003);
    display(q);

    printf("Student with ID %d is served.\n", dequeue(&q));
    display(q);
    return 0;
}

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

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

int dequeue(struct CafeteriaQueue *q) {
    int x = -1;
    if (q->front == q->rear) {
        printf("Queue is empty\n");
```

```
    } else {
        q->front++;
        x = q->Q[q->front];
    }
    return x;
}

void display(struct CafeteriaQueue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
    } else {
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Student ID: %d\n", q.Q[i]);
        }
    }
}

//4.

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

struct HelpDeskQueue {
    int size;
    int front;
    int rear;
    int *Q;
};

void create(struct HelpDeskQueue *, int);
void enqueue(struct HelpDeskQueue *, int);
```

```
int dequeue(struct HelpDeskQueue *);
void display(struct HelpDeskQueue);

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

   enqueue(&q, 1001);
   enqueue(&q, 1002);
   enqueue(&q, 1003);
   display(q);

   printf("Student with ID %d has been helped.\n", dequeue(&q));
   display(q);
   return 0;
}

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

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

```
int dequeue(struct HelpDeskQueue *q) {
    int x = -1;
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        x = q->Q[q->front];
    }
    return x;
}

void display(struct HelpDeskQueue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
    } else {
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Student ID: %d\n", q.Q[i]);
        }
    }
}

//5.

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

struct ExamQueue {
    int size;
    int front;
    int rear;
```

```
    int *Q;
};

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

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

    enqueue(&q, 1001);
    enqueue(&q, 1002);
    enqueue(&q, 1003);
    display(q);

    printf("Student with ID %d registered for the exam.\n", dequeue(&q));
    display(q);
    return 0;
}

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

void enqueue(struct ExamQueue *q, int student_id) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
```

```
    } else {
        q->rear++;
        q->Q[q->rear] = student_id;
    }
}

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

void display(struct ExamQueue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
    } else {
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Student ID: %d\n", q.Q[i]);
        }
    }
}

//6.

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

```
struct BusQueue {
   int size;
   int front;
   int rear;
   int *Q;
};

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

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

   enqueue(&q, 1001);
   enqueue(&q, 1002);
   enqueue(&q, 1003);
   display(q);

   printf("Student with ID %d boarded the bus.\n", dequeue(&q));
   display(q);
   return 0;
}

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

```
}

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

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

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

```
//7.

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

struct CounselingQueue {
    int size;
    int front;
    int rear;
    char **Q;
};

void create(struct CounselingQueue *, int);
void enqueue(struct CounselingQueue *, char*);
char* dequeue(struct CounselingQueue *);
void display(struct CounselingQueue);

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

    enqueue(&q, "Alice");
    enqueue(&q, "Bob");
    enqueue(&q, "Charlie");
    display(q);
    printf("Additional display to show the queue:\n");
    display(q);

    printf("Student %s has been helped.\n", dequeue(&q));
```

```
    display(q);
    return 0;
}

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

void enqueue(struct CounselingQueue *q, char *name) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc(strlen(name) + 1);
        strcpy(q->Q[q->rear], name);
    }
}

char* dequeue(struct CounselingQueue *q) {
    char *name = NULL;
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        name = q->Q[q->front];
    }
    return name;
}
```

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

//8.

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

struct SportsEventQueue {
    int size;
    int front;
    int rear;
    char **Q;
};

void create(struct SportsEventQueue *, int);
void enqueue(struct SportsEventQueue *, char*);
char* dequeue(struct SportsEventQueue *);
void display(struct SportsEventQueue);

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

```
    enqueue(&q, "John");
    enqueue(&q, "Emma");
    enqueue(&q, "Sophia");
    display(q);
    printf("Additional display to show the queue:\n");
    display(q);

    printf("Student %s registered for the sports event.\n", dequeue(&q));
    display(q);
    return 0;
}

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

void enqueue(struct SportsEventQueue *q, char *name) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc(strlen(name) + 1);
        strcpy(q->Q[q->rear], name);
    }
}

char* dequeue(struct SportsEventQueue *q) {
    char *name = NULL;
```

```
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        name = q->Q[q->front];
    }
    return name;
}

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

//9.

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

struct LabQueue {
    int size;
    int front;
    int rear;
    char **Q;
};
```

```
void create(struct LabQueue *, int);
void enqueue(struct LabQueue *, char*);
char* dequeue(struct LabQueue *);
void display(struct LabQueue);

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

    enqueue(&q, "Lucas");
    enqueue(&q, "Mia");
    enqueue(&q, "Olivia");
    display(q);
    printf("Additional display to show the queue:\n");
    display(q);

    printf("Student %s checked out the lab equipment.\n", dequeue(&q));
    display(q);
    return 0;
}

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

void enqueue(struct LabQueue *q, char *name) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
```

```
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc(strlen(name) + 1);
        strcpy(q->Q[q->rear], name);
    }
}

char* dequeue(struct LabQueue *q) {
    char *name = NULL;
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        name = q->Q[q->front];
    }
    return name;
}

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

//10.

#include <stdio.h>
```

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

struct ParentQueue {
   int size;
   int front;
   int rear;
   char **Q;
};

void create(struct ParentQueue *, int);
void enqueue(struct ParentQueue *, char*);
char* dequeue(struct ParentQueue *);
void display(struct ParentQueue);

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

   enqueue(&q, "Mr. Smith");
   enqueue(&q, "Mrs. Johnson");
   enqueue(&q, "Mr. Lee");
   display(q);
   printf("Additional display to show the queue:\n");
   display(q);

   printf("Parent %s met the teacher.\n", dequeue(&q));
   display(q);
   return 0;
}
```

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

void enqueue(struct ParentQueue *q, char *name) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc(strlen(name) + 1);
        strcpy(q->Q[q->rear], name);
    }
}

char* dequeue(struct ParentQueue *q) {
    char *name = NULL;
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        q->front++;
        name = q->Q[q->front];
    }
    return name;
}

void display(struct ParentQueue q) {
    if (q.front == q.rear) {
        printf("Queue is empty\n");
    } else {
```

```
        for (int i = q.front + 1; i <= q.rear; i++) {
            printf("Parent: %s\n", q.Q[i]);
        }
    }
}


//1.

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

struct SensorData {
    int timestamp;
    float temperature;
    float pressure;
    struct SensorData *next;
};

struct SensorQueue {
    struct SensorData *front, *rear;
};

void enqueue(struct SensorQueue *q, int timestamp, float temperature, float pressure);
struct SensorData* dequeue(struct SensorQueue *q);
void display(struct SensorQueue *q);
struct SensorData* search(struct SensorQueue *q, int timestamp);

int main() {
    struct SensorQueue q = {NULL, NULL};
```

```
    enqueue(&q, 1, 25.5, 101.2);
    enqueue(&q, 2, 26.1, 101.5);
    enqueue(&q, 3, 27.0, 101.8);

    display(&q);

    struct SensorData *data = search(&q, 2);
    if (data) {
        printf("Found sensor data at timestamp %d: %.2f, %.2f\n", data->timestamp, data->temperature,
data->pressure);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct SensorQueue *q, int timestamp, float temperature, float pressure) {
    struct SensorData *newData = (struct SensorData*)malloc(sizeof(struct SensorData));
    newData->timestamp = timestamp;
    newData->temperature = temperature;
    newData->pressure = pressure;
    newData->next = NULL;

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

```
}

struct SensorData* dequeue(struct SensorQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct SensorData *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct SensorQueue *q) {
    struct SensorData *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("Timestamp: %d, Temperature: %.2f, Pressure: %.2f\n", temp->timestamp, temp->temperature, temp->pressure);
        temp = temp->next;
    }
}
```

```
struct SensorData* search(struct SensorQueue *q, int timestamp) {
    struct SensorData *temp = q->front;
    while (temp != NULL) {
        if (temp->timestamp == timestamp) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

//2.

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

struct Task {
    int task_id;
    int priority;
    int exec_time;
    struct Task *next;
};

struct TaskQueue {
    struct Task *front, *rear;
};

void enqueue(struct TaskQueue *q, int task_id, int priority, int exec_time);
struct Task* dequeue(struct TaskQueue *q);
void display(struct TaskQueue *q);
```

```
struct Task* search(struct TaskQueue *q, int priority);

int main() {
    struct TaskQueue q = {NULL, NULL};

    enqueue(&q, 1, 5, 100);
    enqueue(&q, 2, 3, 200);
    enqueue(&q, 3, 7, 50);

    display(&q);

    struct Task *task = search(&q, 3);
    if (task) {
        printf("Found task with priority %d: ID=%d, Exec Time=%d\n", task->priority, task->task_id, task->exec_time);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct TaskQueue *q, int task_id, int priority, int exec_time) {
    struct Task *newTask = (struct Task*)malloc(sizeof(struct Task));
    newTask->task_id = task_id;
    newTask->priority = priority;
    newTask->exec_time = exec_time;
    newTask->next = NULL;

    if (q->rear == NULL) {
```

```
        q->front = q->rear = newTask;
    } else {
        q->rear->next = newTask;
        q->rear = newTask;
    }
}

struct Task* dequeue(struct TaskQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct Task *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct TaskQueue *q) {
    struct Task *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
```

```
        printf("Task ID: %d, Priority: %d, Execution Time: %d\n", temp->task_id, temp->priority, temp->exec_time);
        temp = temp->next;
    }
}

struct Task* search(struct TaskQueue *q, int priority) {
    struct Task *temp = q->front;
    while (temp != NULL) {
        if (temp->priority == priority) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

//3.

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

struct Interrupt {
    int irq_id;
    int priority;
    void (*handler)(void);
    struct Interrupt *next;
};

struct IRQQueue {
    struct Interrupt *front, *rear;
```

```
};

void enqueue(struct IRQQueue *q, int irq_id, int priority, void (*handler)(void));
struct Interrupt* dequeue(struct IRQQueue *q);
void display(struct IRQQueue *q);
struct Interrupt* search(struct IRQQueue *q, int irq_id);

void example_handler(void) {
    printf("Interrupt handler invoked!\n");
}

int main() {
    struct IRQQueue q = {NULL, NULL};

    enqueue(&q, 1, 5, example_handler);
    enqueue(&q, 2, 3, example_handler);
    enqueue(&q, 3, 7, example_handler);

    display(&q);

    struct Interrupt *irq = search(&q, 3);
    if (irq) {
        printf("Found IRQ with ID %d, priority %d\n", irq->irq_id, irq->priority);
        irq->handler();
    }

    dequeue(&q);
    display(&q);

    return 0;
}
```

```
void enqueue(struct IRQQueue *q, int irq_id, int priority, void (*handler)(void)) {
    struct Interrupt *newInterrupt = (struct Interrupt*)malloc(sizeof(struct Interrupt));
    newInterrupt->irq_id = irq_id;
    newInterrupt->priority = priority;
    newInterrupt->handler = handler;
    newInterrupt->next = NULL;

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

struct Interrupt* dequeue(struct IRQQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct Interrupt *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}
```

```
void display(struct IRQQueue *q) {
    struct Interrupt *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("IRQ ID: %d, Priority: %d\n", temp->irq_id, temp->priority);
        temp = temp->next;
    }
}

struct Interrupt* search(struct IRQQueue *q, int irq_id) {
    struct Interrupt *temp = q->front;
    while (temp != NULL) {
        if (temp->irq_id == irq_id) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}
```

//4.

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

```
struct Message {
    int sender_id;
    int receiver_id;
    char payload[100];
    struct Message *next;
};

struct MessageQueue {
    struct Message *front, *rear;
};

void enqueue(struct MessageQueue *q, int sender_id, int receiver_id, const char *payload);
struct Message* dequeue(struct MessageQueue *q);
void display(struct MessageQueue *q);
struct Message* search(struct MessageQueue *q, int sender_id);

int main() {
    struct MessageQueue q = {NULL, NULL};

    enqueue(&q, 1, 2, "Hello");
    enqueue(&q, 2, 3, "How are you?");
    enqueue(&q, 1, 3, "Goodbye");

    display(&q);

    struct Message *msg = search(&q, 1);
    if (msg) {
        printf("Found message from sender %d: %s\n", msg->sender_id, msg->payload);
    }
```

```
    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct MessageQueue *q, int sender_id, int receiver_id, const char *payload) {
    struct Message *newMessage = (struct Message*)malloc(sizeof(struct Message));
    newMessage->sender_id = sender_id;
    newMessage->receiver_id = receiver_id;
    strcpy(newMessage->payload, payload);
    newMessage->next = NULL;

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

struct Message* dequeue(struct MessageQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct Message *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
```

```
    }

    free(temp);
    return temp;
}

void display(struct MessageQueue *q) {
    struct Message *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("Sender: %d, Receiver: %d, Message: %s\n", temp->sender_id, temp->receiver_id, temp->payload);
        temp = temp->next;
    }
}

struct Message* search(struct MessageQueue *q, int sender_id) {
    struct Message *temp = q->front;
    while (temp != NULL) {
        if (temp->sender_id == sender_id) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}
```

```
//5.

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

struct LogEntry {
    int timestamp;
    char event_type[50];
    char description[100];
    struct LogEntry *next;
};

struct LogQueue {
    struct LogEntry *front, *rear;
};

void enqueue(struct LogQueue *q, int timestamp, const char *event_type, const char *description);
struct LogEntry* dequeue(struct LogQueue *q);
void display(struct LogQueue *q);
struct LogEntry* search(struct LogQueue *q, const char *event_type);

int main() {
    struct LogQueue q = {NULL, NULL};

    enqueue(&q, 1, "Error", "Sensor failure");
    enqueue(&q, 2, "Info", "Device started");
    enqueue(&q, 3, "Warning", "Battery low");

    display(&q);
```

```
    struct LogEntry *log = search(&q, "Info");
    if (log) {
        printf("Found log entry: %s - %s\n", log->event_type, log->description);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct LogQueue *q, int timestamp, const char *event_type, const char *description) {
    struct LogEntry *newLog = (struct LogEntry*)malloc(sizeof(struct LogEntry));
    newLog->timestamp = timestamp;
    strcpy(newLog->event_type, event_type);
    strcpy(newLog->description, description);
    newLog->next = NULL;

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

struct LogEntry* dequeue(struct LogQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }
```

```
    struct LogEntry *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct LogQueue *q) {
    struct LogEntry *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("Timestamp: %d, Event: %s, Description: %s\n", temp->timestamp, temp->event_type,
temp->description);
        temp = temp->next;
    }
}

struct LogEntry* search(struct LogQueue *q, const char *event_type) {
    struct LogEntry *temp = q->front;
    while (temp != NULL) {
        if (strcmp(temp->event_type, event_type) == 0) {
            return temp;
        }
```

```
        temp = temp->next;
    }
    return NULL;
}

//6.

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

struct Packet {
    char source_ip[16];
    char dest_ip[16];
    char payload[100];
    struct Packet *next;
};

struct PacketQueue {
    struct Packet *front, *rear;
};

void enqueue(struct PacketQueue *q, const char *source_ip, const char *dest_ip, const char
*payload);
struct Packet* dequeue(struct PacketQueue *q);
void display(struct PacketQueue *q);
struct Packet* search(struct PacketQueue *q, const char *source_ip);

int main() {
    struct PacketQueue q = {NULL, NULL};
```

```
    enqueue(&q, "192.168.1.1", "192.168.1.2", "Hello");
    enqueue(&q, "192.168.1.3", "192.168.1.4", "Data");
    enqueue(&q, "192.168.1.5", "192.168.1.6", "Goodbye");

    display(&q);

    struct Packet *pkt = search(&q, "192.168.1.3");
    if (pkt) {
        printf("Found packet from %s to %s: %s\n", pkt->source_ip, pkt->dest_ip, pkt->payload);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct PacketQueue *q, const char *source_ip, const char *dest_ip, const char
*payload) {
    struct Packet *newPacket = (struct Packet*)malloc(sizeof(struct Packet));
    strcpy(newPacket->source_ip, source_ip);
    strcpy(newPacket->dest_ip, dest_ip);
    strcpy(newPacket->payload, payload);
    newPacket->next = NULL;

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

```
}

struct Packet* dequeue(struct PacketQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct Packet *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct PacketQueue *q) {
    struct Packet *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("Source IP: %s, Destination IP: %s, Payload: %s\n", temp->source_ip, temp->dest_ip,
temp->payload);
        temp = temp->next;
    }
}
```

```
struct Packet* search(struct PacketQueue *q, const char *source_ip) {
    struct Packet *temp = q->front;
    while (temp != NULL) {
        if (strcmp(temp->source_ip,  source_ip) == 0) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

//7.

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

struct FirmwareUpdate {
    int version;
    char release_notes[100];
    char file_path[100];
    struct FirmwareUpdate *next;
};

struct FirmwareQueue {
    struct FirmwareUpdate *front, *rear;
};

void enqueue(struct FirmwareQueue *q, int version, const char *release_notes, const char
*file_path);
```

```
struct FirmwareUpdate* dequeue(struct FirmwareQueue *q);
void display(struct FirmwareQueue *q);
struct FirmwareUpdate* search(struct FirmwareQueue *q, int version);

int main() {
    struct FirmwareQueue q = {NULL, NULL};

    enqueue(&q, 1, "Initial release", "/path/to/firmware_v1");
    enqueue(&q, 2, "Bug fixes", "/path/to/firmware_v2");
    enqueue(&q, 3, "Security patch", "/path/to/firmware_v3");

    display(&q);

    struct FirmwareUpdate *update = search(&q, 2);
    if (update) {
        printf("Found firmware update version %d: %s\n", update->version, update->release_notes);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct FirmwareQueue *q, int version, const char *release_notes, const char
*file_path) {
    struct FirmwareUpdate *newUpdate = (struct FirmwareUpdate*)malloc(sizeof(struct
FirmwareUpdate));
    newUpdate->version = version;
    strcpy(newUpdate->release_notes, release_notes);
    strcpy(newUpdate->file_path, file_path);
    newUpdate->next = NULL;
```

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

struct FirmwareUpdate* dequeue(struct FirmwareQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct FirmwareUpdate *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct FirmwareQueue *q) {
    struct FirmwareUpdate *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }
```

```
    while (temp != NULL) {
        printf("Version: %d, Release Notes: %s, File Path: %s\n", temp->version, temp->release_notes,
temp->file_path);
        temp = temp->next;
    }
}

struct FirmwareUpdate* search(struct FirmwareQueue *q, int version) {
    struct FirmwareUpdate *temp = q->front;
    while (temp != NULL) {
        if (temp->version == version) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

//8.

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

struct PowerEvent {
    char event_type[50];
    int timestamp;
    char action[100];
    struct PowerEvent *next;
};
```

```
struct PowerEventQueue {
    struct PowerEvent *front, *rear;
};

void enqueue(struct PowerEventQueue *q, const char *event_type, int timestamp, const char *action);
struct PowerEvent* dequeue(struct PowerEventQueue *q);
void display(struct PowerEventQueue *q);
struct PowerEvent* search(struct PowerEventQueue *q, const char *event_type);

int main() {
    struct PowerEventQueue q = {NULL, NULL};

    enqueue(&q, "Power On", 1, "Start device");
    enqueue(&q, "Sleep", 2, "Put device in sleep mode");
    enqueue(&q, "Power Off", 3, "Turn off device");

    display(&q);

    struct PowerEvent *event = search(&q, "Sleep");
    if (event) {
        printf("Found event: %s - %s\n", event->event_type, event->action);
    }

    dequeue(&q);
    display(&q);

    return 0;
}
```

```
void enqueue(struct PowerEventQueue *q, const char *event_type, int timestamp, const char *action) {
    struct PowerEvent *newEvent = (struct PowerEvent*)malloc(sizeof(struct PowerEvent));
    strcpy(newEvent->event_type, event_type);
    newEvent->timestamp = timestamp;
    strcpy(newEvent->action, action);
    newEvent->next = NULL;

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

struct PowerEvent* dequeue(struct PowerEventQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct PowerEvent *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}
```

```
void display(struct PowerEventQueue *q) {
    struct PowerEvent *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("Timestamp: %d, Event: %s, Action: %s\n", temp->timestamp, temp->event_type, temp->action);
        temp = temp->next;
    }
}

struct PowerEvent* search(struct PowerEventQueue *q, const char *event_type) {
    struct PowerEvent *temp = q->front;
    while (temp != NULL) {
        if (strcmp(temp->event_type, event_type) == 0) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

//9.

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

```
struct Command {
   int id;
   char type[50];
   char parameters[100];
   struct Command *next;
};

struct CommandQueue {
   struct Command *front, *rear;
};

void enqueue(struct CommandQueue *q, int id, const char *type, const char *parameters);
struct Command* dequeue(struct CommandQueue *q);
void display(struct CommandQueue *q);
struct Command* search(struct CommandQueue *q, const char *type);

int main() {
   struct CommandQueue q = {NULL, NULL};

   enqueue(&q, 1, "Read", "Sensor1");
   enqueue(&q, 2, "Write", "Sensor2");
   enqueue(&q, 3, "Execute", "TaskA");

   display(&q);

   struct Command *cmd = search(&q, "Write");
   if (cmd) {
      printf("Found command: %d, Type: %s, Parameters: %s\n", cmd->id, cmd->type, cmd-
>parameters);
   }
```

```
    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct CommandQueue *q, int id, const char *type, const char *parameters) {
    struct Command *newCmd = (struct Command*)malloc(sizeof(struct Command));
    newCmd->id = id;
    strcpy(newCmd->type, type);
    strcpy(newCmd->parameters, parameters);
    newCmd->next = NULL;

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

struct Command* dequeue(struct CommandQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct Command *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
```

```
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct CommandQueue *q) {
    struct Command *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("ID: %d, Type: %s, Parameters: %s\n", temp->id, temp->type, temp->parameters);
        temp = temp->next;
    }
}

struct Command* search(struct CommandQueue *q, const char *type) {
    struct Command *temp = q->front;
    while (temp != NULL) {
        if (strcmp(temp->type, type) == 0) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}
```

```
//10.

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

struct AudioSample {
    int timestamp;
    char data[100]; // Example audio data
    struct AudioSample *next;
};

struct AudioQueue {
    struct AudioSample *front, *rear;
};

void enqueue(struct AudioQueue *q, int timestamp, const char *data);
struct AudioSample* dequeue(struct AudioQueue *q);
void display(struct AudioQueue *q);
struct AudioSample* search(struct AudioQueue *q, int timestamp);

int main() {
    struct AudioQueue q = {NULL, NULL};

    enqueue(&q, 1, "Sample1");
    enqueue(&q, 2, "Sample2");
    enqueue(&q, 3, "Sample3");

    display(&q);

    struct AudioSample *sample = search(&q, 2);
    if (sample) {
```

```
        printf("Found sample with timestamp %d: %s\n", sample->timestamp, sample->data);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct AudioQueue *q, int timestamp, const char *data) {
    struct AudioSample *newSample = (struct AudioSample*)malloc(sizeof(struct AudioSample));
    newSample->timestamp = timestamp;
    strcpy(newSample->data, data);
    newSample->next = NULL;

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

struct AudioSample* dequeue(struct AudioQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct AudioSample *temp = q->front;
    q->front = q->front->next;
```

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

    free(temp);
    return temp;
}

void display(struct AudioQueue *q) {
    struct AudioSample *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("Timestamp: %d, Data: %s\n", temp->timestamp, temp->data);
        temp = temp->next;
    }
}

struct AudioSample* search(struct AudioQueue *q, int timestamp) {
    struct AudioSample *temp = q->front;
    while (temp != NULL) {
        if (temp->timestamp == timestamp) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}
```

```
//11.

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

struct Event {
    int id;
    char type[50];
    char data[100];
    struct Event *next;
};

struct EventQueue {
    struct Event *front, *rear;
};

void enqueue(struct EventQueue *q, int id, const char *type, const char *data);
struct Event* dequeue(struct EventQueue *q);
void display(struct EventQueue *q);
struct Event* search(struct EventQueue *q, int id);

int main() {
    struct EventQueue q = {NULL, NULL};

    enqueue(&q, 1, "Button Press", "Button 1");
    enqueue(&q, 2, "Sensor Trigger", "Temperature exceeded threshold");
    enqueue(&q, 3, "Timeout", "Process timeout");

    display(&q);
```

```
    struct Event *event = search(&q, 2);
    if (event) {
        printf("Found event: %d, Type: %s, Data: %s\n", event->id, event->type, event->data);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct EventQueue *q, int id, const char *type, const char *data) {
    struct Event *newEvent = (struct Event*)malloc(sizeof(struct Event));
    newEvent->id = id;
    strcpy(newEvent->type, type);
    strcpy(newEvent->data, data);
    newEvent->next = NULL;

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

struct Event* dequeue(struct EventQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }
```

```
    struct Event *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct EventQueue *q) {
    struct Event *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("ID: %d, Type: %s, Data: %s\n", temp->id, temp->type, temp->data);
        temp = temp->next;
    }
}

struct Event* search(struct EventQueue *q, int id) {
    struct Event *temp = q->front;
    while (temp != NULL) {
        if (temp->id == id) {
            return temp;
        }
        temp = temp->next;
```

```
    }
    return NULL;
}

//12.

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

struct GUIEvent {
    int id;
    char type[50];
    int x, y; // Coordinates
    int timestamp;
    struct GUIEvent *next;
};

struct GUIEventQueue {
    struct GUIEvent *front, *rear;
};

void enqueue(struct GUIEventQueue *q, int id, const char *type, int x, int y, int timestamp);
struct GUIEvent* dequeue(struct GUIEventQueue *q);
void display(struct GUIEventQueue *q);
struct GUIEvent* search(struct GUIEventQueue *q, int id);

int main() {
    struct GUIEventQueue q = {NULL, NULL};

    enqueue(&q, 1, "Button Click", 100, 150, 1000);
    enqueue(&q, 2, "Screen Touch", 200, 250, 2000);
```

```
    enqueue(&q, 3, "Swipe", 300, 350, 3000);

    display(&q);

    struct GUIEvent *event = search(&q, 2);
    if (event) {
        printf("Found event: %d, Type: %s, Coordinates: (%d, %d)\n", event->id, event->type, event->x,
event->y);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct GUIEventQueue *q, int id, const char *type, int x, int y, int timestamp) {
    struct GUIEvent *newEvent = (struct GUIEvent*)malloc(sizeof(struct GUIEvent));
    newEvent->id = id;
    strcpy(newEvent->type, type);
    newEvent->x = x;
    newEvent->y = y;
    newEvent->timestamp = timestamp;
    newEvent->next = NULL;

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

```
}

struct GUIEvent* dequeue(struct GUIEventQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct GUIEvent *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct GUIEventQueue *q) {
    struct GUIEvent *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("ID: %d, Type: %s, Coordinates: (%d, %d), Timestamp: %d\n", temp->id, temp->type,
temp->x, temp->y, temp->timestamp);
        temp = temp->next;
    }
}
```

```
struct GUIEvent* search(struct GUIEventQueue *q, int id) {
    struct GUIEvent *temp = q->front;
    while (temp != NULL) {
        if (temp->id == id) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

//13.

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

struct DataChunk {
    char data[100]; // Example data
    int length;
    struct DataChunk *next;
};

struct SerialQueue {
    struct DataChunk *front, *rear;
};

void enqueue(struct SerialQueue *q, const char *data, int length);
struct DataChunk* dequeue(struct SerialQueue *q);
void display(struct SerialQueue *q);
```

```
struct DataChunk* search(struct SerialQueue *q, const char *data);

int main() {
    struct SerialQueue q = {NULL, NULL};

    enqueue(&q, "Hello", 5);
    enqueue(&q, "World", 5);
    enqueue(&q, "Test", 4);

    display(&q);

    struct DataChunk *chunk = search(&q, "World");
    if (chunk) {
        printf("Found data: %s\n", chunk->data);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct SerialQueue *q, const char *data, int length) {
    struct DataChunk *newChunk = (struct DataChunk*)malloc(sizeof(struct DataChunk));
    strcpy(newChunk->data, data);
    newChunk->length = length;
    newChunk->next = NULL;

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

```
        q->rear->next = newChunk;
        q->rear = newChunk;
    }
}

struct DataChunk* dequeue(struct SerialQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct DataChunk *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct SerialQueue *q) {
    struct DataChunk *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("Data: %s, Length: %d\n", temp->data, temp->length);
        temp = temp->next;
```

```
    }
}

struct DataChunk* search(struct SerialQueue *q, const char *data) {
    struct DataChunk *temp = q->front;
    while (temp != NULL) {
        if (strcmp(temp->data, data) == 0) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

//14.

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

struct CANMessage {
    int id;
    int length;
    char payload[100];
    struct CANMessage *next;
};

struct CANQueue {
    struct CANMessage *front, *rear;
};
```

```
void enqueue(struct CANQueue *q, int id, int length, const char *payload);
struct CANMessage* dequeue(struct CANQueue *q);
void display(struct CANQueue *q);
struct CANMessage* search(struct CANQueue *q, int id);

int main() {
    struct CANQueue q = {NULL, NULL};

    enqueue(&q, 1, 5, "Message1");
    enqueue(&q, 2, 6, "Message2");
    enqueue(&q, 3, 7, "Message3");

    display(&q);

    struct CANMessage *msg = search(&q, 2);
    if (msg) {
        printf("Found message ID: %d, Payload: %s\n", msg->id, msg->payload);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct CANQueue *q, int id, int length, const char *payload) {
    struct CANMessage *newMsg = (struct CANMessage*)malloc(sizeof(struct CANMessage));
    newMsg->id = id;
    newMsg->length = length;
    strcpy(newMsg->payload, payload);
    newMsg->next = NULL;
```

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

struct CANMessage* dequeue(struct CANQueue *q) {
    if (q->front == NULL) {
        printf("Queue is empty\n");
        return NULL;
    }

    struct CANMessage *temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

void display(struct CANQueue *q) {
    struct CANMessage *temp = q->front;
    if (!temp) {
        printf("Queue is empty\n");
        return;
    }
```

```
    while (temp != NULL) {
        printf("ID: %d, Length: %d, Payload: %s\n", temp->id, temp->length, temp->payload);
        temp = temp->next;
    }
}

struct CANMessage* search(struct CANQueue *q, int id) {
    struct CANMessage *temp = q->front;
    while (temp != NULL) {
        if (temp->id == id) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

//15.

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

struct MLData {
    int id;
    char features[100];
    char metadata[100];
    struct MLData *next;
};
```

```
struct MLQueue {
    struct MLData *front, *rear;
};

void enqueue(struct MLQueue *q, int id, const char *features, const char *metadata);
struct MLData* dequeue(struct MLQueue *q);
void display(struct MLQueue *q);
struct MLData* search(struct MLQueue *q, int id);

int main() {
    struct MLQueue q = {NULL, NULL};

    enqueue(&q, 1, "Feature1", "Metadata1");
    enqueue(&q, 2, "Feature2", "Metadata2");
    enqueue(&q, 3, "Feature3", "Metadata3");

    display(&q);

    struct MLData *data = search(&q, 2);
    if (data) {
        printf("Found data ID: %d, Features: %s, Metadata: %s\n", data->id, data->features, data->metadata);
    }

    dequeue(&q);
    display(&q);

    return 0;
}

void enqueue(struct MLQueue *q, int id, const char *features, const char *metadata) {
```

```
    struct MLData *newData = (struct MLData*)malloc(sizeof(struct  MLData));
    newData->id = id;
    strcpy(newData->features,  features);
    strcpy(newData->metadata,  metadata);
    newData->next = NULL;

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

struct MLData* dequeue(struct  MLQueue  *q) {
    if (q->front  == NULL) {
        printf("Queue  is empty\n");
        return NULL;
    }

    struct MLData *temp = q->front;
    q->front = q->front->next;
    if (q->front  == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return temp;
}

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

```
    struct MLData *temp = q->front;
    if (!temp) {
        printf("Queue  is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("ID: %d, Features: %s, Metadata: %s\n", temp->id, temp->features, temp->metadata);
        temp = temp->next;
    }
}

struct MLData* search(struct MLQueue *q, int id) {
    struct MLData *temp = q->front;
    while (temp != NULL) {
        if (temp->id == id) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}
```