Assignment -22

Variable, Static, Const, and Switch Case

1. **Question 1:** Write a C program that declares a static variable and a const variable within a function. The program should increment the static variable each time the function is called and use a switch case to check the value of the const variable. The function should handle at least three different cases for the const variable and demonstrate the persistence of the static variable across multiple calls.

Sol:

```
# include <stdio.h>

void demoFun(){

   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(){
    demoFun();
    demoFun();
    demoFun();
}
```

2. **Question 2:** Create a C program where a static variable is used to keep track of the number of times a function has been called. Implement a switch case to print a different message based on the number of times the function has been invoked (e.g., first call, second call, more than two calls). Ensure that a const variable is used to define a maximum call limit and terminate further calls once the limit is reached.

Sol:

```
#include <stdio.h>
// Function declaration
void trackFunctionCalls();
int main() {
    for (int i = 0; i < 5; i++) {  // Simulate multiple function calls
        trackFunctionCalls();
    }
    return 0;
}
void trackFunctionCalls() {
    static int callCount = 0;  // Static variable to keep track of the number of calls
    const int maxCalls = 3;    // Const variable to set the maximum call limit
```

```
    if (callCount >= maxCalls) {
        printf("Call limit reached. No further calls are allowed.\n");
        return;
    }
    callCount++;
    switch (callCount) {
        case 1:
            printf("This is the first call.\n");
            break;
        case 2:
            printf("This is the second call.\n");
            break;
        default:
            printf("This is call number %d.\n", callCount);
            break;
    }
}
```

3. **Question 3:** Develop a C program that utilizes a static array inside a function to store values across multiple calls. Use a const variable to define the size of the array. Implement a switch case to perform different operations on the array elements (e.g., add, subtract, multiply) based on user input. Ensure the array values persist between function calls.

Sol:

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

```
void operateOnArray(int operation);
int main() {
    operateOnArray(1);  // Add to array elements
    operateOnArray(2);  // Subtract from array elements
    operateOnArray(3);  // Multiply array elements
    operateOnArray(1);  // Add again to show persistence
    return 0;
}
void operateOnArray(int operation) {
    const int size = 3;  // Size of the array
    static int array[3] = {2, 4, 6};  // Static array to retain values
    printf("Array before operation: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
  switch (operation) {
        case 1:
            for (int i = 0; i < size; i++) {
                array[i] += 1;
            }
            printf("Added 1 to each element.\n");
            break;
```

```
        case 2:
            for (int i = 0; i < size; i++) {
                array[i] -= 1;
            }
            printf("Subtracted 1 from each element.\n");
            break;
        case 3:
            for (int i = 0; i < size; i++) {
                array[i] *= 2;
            }
            printf("Multiplied each element by 2.\n");
            break;
        default:
            printf("Invalid operation.\n");
            break;
    }
    printf("Array after operation: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }
    printf("\n\n");
}
```

4. **Question 4:** Write a program that demonstrates the difference between const and static variables. Use a static variable to count the number of times a specific switch case is

executed, and a const variable to define a threshold value for triggering a specific case. The program should execute different actions based on the value of the static counter compared to the const threshold.

Sol:

```
#include <stdio.h>

void checkThreshold(int input);

int main() {
    for (int i = 0; i < 7; i++) {  // Loop to simulate multiple inputs
        checkThreshold(i % 3);     // Cycling through values 0, 1, and 2
    }
    return 0;
}

void checkThreshold(int input) {
    static int counter = 0;  // Static variable to count switch case executions
    const int threshold = 3; // Const variable to define the threshold value
    switch (input) {
        case 0:
            printf("Case 0 executed.\n");
            counter++;
            break;
        case 1:
            printf("Case 1 executed.\n");
            counter++;
            break;
        case 2:
```

```
            printf("Case 2 executed.\n");
            counter++;
            break;
        default:
            printf("Invalid input.\n");
            break;
    }
    printf("Static counter value: %d\n", counter);
    if (counter >= threshold) {
        printf("Threshold reached! Taking specific action.\n");
    } else {
        printf("Threshold not yet reached.\n");
    }
    printf("\n");
}
```

5. **Question 5:** Create a C program with a static counter and a const limit. The program should include a switch case to print different messages based on the value of the counter. After every 5 calls, reset the counter using the const limit. The program should also demonstrate the immutability of the const variable by attempting to modify it and showing the compilation error.

Sol:

```
#include <stdio.h>

#define LIMIT 5  // Define the constant limit

void incrementCounter() {
    static int counter = 0;  // Static variable to retain its value between function calls
```

```
// Increment the counter
counter++;
// Switch-case to print messages based on the counter's value
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;  // Reset the counter after 5 calls
        break;
    default:
        printf("Counter reset to 0\n");
        break;
```

```
    }

    // Attempt to modify the const LIMIT variable (this will cause a compilation error)

    // LIMIT = 10;  // Uncommenting this line will cause a compilation error

}

int main() {

    // Call incrementCounter multiple times

    for (int i = 0; i < 10; i++) {

        incrementCounter();

    }

    return 0;

}
```

Looping Statements, Pointers, Const with Pointers, Functions

1. **Question 1:** Write a C program that demonstrates the use of both single and double pointers. Implement a function that uses a for loop to initialize an array and a second function that modifies the array elements using a double pointer. Use the const keyword to prevent modification of the array elements in one of the functions.

Sol:

```
#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; // Double the value at each index
    }
}
```

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

Question 2: Develop a program that reads a matrix from the user and uses a function to transpose the matrix. The function should use a double pointer to manipulate the matrix. Demonstrate both call by value and call by reference in the program. Use a const pointer to ensure the original matrix is not modified during the transpose operation.

Sol:

```
#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");
    }
}
```

Question 3: Create a C program that uses a single pointer to dynamically allocate memory for an array. Write a function to initialize the array using a while loop, and another function to print the array. Use a const pointer to ensure the printing function does not modify the array.

Sol:

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

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

int main() {
    const int SIZE = 5;
    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");
}
```

Question 4: Write a program that demonstrates the use of double pointers to swap two arrays. Implement functions using both call by value and call by reference. Use a for loop to print the swapped arrays and apply the const keyword appropriately to ensure no modification occurs in certain operations.

Sol:

```
#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[] = {1, 2, 3, 4, 5};
    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");
}
```

Question 5: Develop a C program that demonstrates the application of const with pointers. Create a function to read a string from the user and another function to count the frequency of each character using a do-while loop. Use a const pointer to ensure the original string is not modified during character frequency calculation.

Sol: #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]);
        }
    }
}
```

Arrays, Structures, Nested Structures, Unions, Nested Unions, Strings, Typedef

1. **Question 1:** Write a C program that uses an array of structures to store information about employees. Each structure should contain a nested structure for the address. Use typedef to simplify the structure definitions. The program should allow the user to enter and display employee information.

Sol:

```
#include <stdio.h>

#define MAX_NAME_LENGTH 50

#define MAX_ADDRESS_LENGTH 100

#define NUM_EMPLOYEES 3  // Number of employees

// Define the Address structure using typedef

typedef struct {
    char street[MAX_ADDRESS_LENGTH];
    char city[MAX_ADDRESS_LENGTH];
    char state[MAX_ADDRESS_LENGTH];
    int zipCode;
} Address;
```

```
// Define the Employee structure using typedef, which contains a nested Address structure
typedef struct {
    char name[MAX_NAME_LENGTH];
    int age;
    float salary;
    Address address;  // Nested structure for address
} Employee;
// Function to take employee information as input
void inputEmployeeData(Employee *emp) {
    printf("Enter employee name: ");
    scanf("%s", emp->name);  // Using scanf to read a string
    printf("Enter employee age: ");
    scanf("%d", &emp->age);
    printf("Enter employee salary: ");
    scanf("%f", &emp->salary);
    // Input for the address
    printf("Enter street address: ");
    scanf(" %[^\n]%*c", emp->address.street);  // Reads full line for address
        printf("Enter city: ");
    scanf(" %[^\n]%*c", emp->address.city);  // Reads full line for city
printf("Enter state: ");
    scanf(" %[^\n]%*c", emp->address.state);  // Reads full line for state
    printf("Enter zip code: ");
```

```
    scanf("%d", &emp->address.zipCode);
}
// Function to display employee information
void displayEmployeeData(Employee emp) {
    printf("\nEmployee Information:\n");
    printf("Name: %s\n", emp.name);
    printf("Age: %d\n", emp.age);
    printf("Salary: %.2f\n", emp.salary);
    printf("Address:\n");
    printf("  Street: %s\n", emp.address.street);
    printf("  City: %s\n", emp.address.city);
    printf("  State: %s\n", emp.address.state);
    printf("  Zip Code: %d\n", emp.address.zipCode);
}
int main() {
    int n = NUM_EMPLOYEES;
    // Create an array of Employee structures
    Employee employees[n];
    // Input and display employee information
    for (int i = 0; i < n; i++) {
        printf("\nEnter details for employee %d:\n", i + 1);
        inputEmployeeData(&employees[i]);  // Take input for each employee
    }
```

```
    // Display employee information
    for (int i = 0; i < n; i++) {
        displayEmployeeData(employees[i]);  // Display details of each employee
    }
    return 0;
}
```

Question 2: Create a program that demonstrates the use of a union to store different types of data. Implement a nested union within a structure and use a typedef to define the structure. Use an array of this structure to store and display information about different data types (e.g., integer, float, string).

Sol:

```
#include <stdio.h>

#define NUM_ITEMS 3  // Number of items in the array

// Define the nested union using typedef
typedef union {
    int intValue;
    float floatValue;
    char strValue[50];
} DataType;  // Union to store an integer, float, or string

// Define the structure using typedef, which contains a nested union
typedef struct {
    int type;        // Type to specify which data is stored
    DataType data;   // Nested union for different data types
} Item;

// Function to input the data for the item
```

```
void inputItemData(Item *item) {
    printf("Enter the type of data (1 for integer, 2 for float, 3 for string): ");
    scanf("%d", &item->type);
    // Input based on the type of data
    if (item->type == 1) {
        printf("Enter an integer: ");
        scanf("%d", &item->data.intValue);
    } else if (item->type == 2) {
        printf("Enter a float: ");
        scanf("%f", &item->data.floatValue);
    } else if (item->type == 3) {
        printf("Enter a string: ");
        scanf(" %[^\n]%*c", item->data.strValue);  // Reads a full line of text
    } else {
        printf("Invalid type\n");
    }
}

// Function to display the data for the item
void displayItemData(Item item) {
    printf("\nItem Type: ");
    if (item.type == 1) {
        printf("Integer\n");
```

```
        printf("Value: %d\n", item.data.intValue);
    } else if (item.type == 2) {
        printf("Float\n");
        printf("Value: %.2f\n", item.data.floatValue);
    } else if (item.type == 3) {
        printf("String\n");
        printf("Value: %s\n", item.data.strValue);
    } else {
        printf("Invalid type\n");
    }
}
int main() {
    Item items[NUM_ITEMS];  // Array to store multiple items
    // Input data for each item
    for (int i = 0; i < NUM_ITEMS; i++) {
        printf("\nEnter data for item %d:\n", i + 1);
        inputItemData(&items[i]);
    }
    // Display the data for each item
    for (int i = 0; i < NUM_ITEMS; i++) {
        displayItemData(items[i]);
    }
 return 0;
```

}

Question 3: Write a C program that uses an array of strings to store names. Implement a structure containing a nested union to store either the length of the string or the reversed string. Use typedef to simplify the structure definition and display the stored information.

Sol:

```
#include <stdio.h>

#include <string.h>

#define MAX_NAMES 3         // Number of names

#define MAX_NAME_LENGTH 50  // Maximum length of each name

// Define the union to store either the length of the string or the reversed string

typedef union {

    int length;          // To store the length of the string

    char reversed[50];    // To store the reversed string

} StringData;

// Define the structure to store the name and the nested union

typedef struct {

    char name[MAX_NAME_LENGTH];  // Store the name as a string

    StringData data;           // Nested union to store length or reversed string

    int isLength;              // Flag to indicate if we are storing the length (1) or reversed string (0)

} NameInfo;

// Function to reverse a string

void reverseString(char *str, char *reversed) {

    int len = strlen(str);

    for (int i = 0; i < len; i++) {

        reversed[i] = str[len - i - 1];
```

```
    }
    reversed[len] = '\0';  // Null-terminate the reversed string
}
// Function to input the name and process the data
void inputNameInfo(NameInfo *info) {
    printf("Enter a name: ");
    scanf("%s", info->name);
    // Ask the user if they want to store the length or the reversed string
    printf("Enter 1 to store length, 0 to store reversed string: ");
    scanf("%d", &info->isLength);
    if (info->isLength == 1) {
        // Store the length of the string
        info->data.length = strlen(info->name);
    } else {
        // Store the reversed string
        reverseString(info->name, info->data.reversed);
    }
}
// Function to display the stored information
void displayNameInfo(NameInfo info) {
    printf("\nName: %s\n", info.name);
    if (info.isLength == 1) {
        printf("Stored as Length: %d\n", info.data.length);
```

```
    } else {
        printf("Stored as Reversed String: %s\n", info.data.reversed);
    }
}
int main() {
    NameInfo names[MAX_NAMES];  // Array to store information about multiple names
    // Input and display the information for each name
    for (int i = 0; i < MAX_NAMES; i++) {
        printf("\nEnter information for name %d:\n", i + 1);
        inputNameInfo(&names[i]);
    }

    // Display the information for each name
    for (int i = 0; i < MAX_NAMES; i++) {
        displayNameInfo(names[i]);
    }
 return 0;
}
```

Question 4: Develop a program that demonstrates the use of nested structures and unions. Create a structure that contains a union, and within the union, define another structure. Use an array to manage multiple instances of this complex structure and typedef to define the structure.

Sol:

```
#include <stdio.h>
#include <string.h>
#define MAX_ITEMS 3  // Number of items
```

```
// Define a structure that will be inside the union
typedef struct {
    char name[50];
    int id;
} ItemDetails;
// Define the union that contains the structure
typedef union {
    int integerData;
    float floatData;
    ItemDetails details;  // Structure inside the union
} DataUnion;
// Define the main structure containing the union
typedef struct {
    char itemType[20];  // Type of item (e.g., "integer", "float", "details")
    DataUnion data;     // Union that holds either an integer, float, or ItemDetails
} ComplexItem;
// Function to input data into ComplexItem
void inputComplexItem(ComplexItem *item) {
    printf("Enter item type (integer, float, details): ");
    scanf("%s", item->itemType);
        // Depending on the item type, store appropriate data in the union
    if (strcmp(item->itemType, "integer") == 0) {
        printf("Enter an integer: ");
```

```
        scanf("%d", &item->data.integerData);
    } else if (strcmp(item->itemType, "float") == 0) {
        printf("Enter a float: ");
        scanf("%f", &item->data.floatData);
    } else if (strcmp(item->itemType, "details") == 0) {
        printf("Enter name: ");
        scanf("%s", item->data.details.name);
        printf("Enter ID: ");
        scanf("%d", &item->data.details.id);
    } else {
        printf("Invalid item type!\n");
    }
}
// Function to display the stored data from ComplexItem
void displayComplexItem(ComplexItem item) {
    printf("\nItem Type: %s\n", item.itemType);
        if (strcmp(item.itemType, "integer") == 0) {
        printf("Stored as Integer: %d\n", item.data.integerData);
    } else if (strcmp(item.itemType, "float") == 0) {
        printf("Stored as Float: %.2f\n", item.data.floatData);
    } else if (strcmp(item.itemType, "details") == 0) {
        printf("Stored as Details: Name = %s, ID = %d\n", item.data.details.name,
item.data.details.id);
    } else {
```

```
        printf("Invalid item type!\n");
    }
}
int main() {
    ComplexItem items[MAX_ITEMS];  // Array to store multiple complex items
    // Input data for each complex item
    for (int i = 0; i < MAX_ITEMS; i++) {
        printf("\nEnter data for item %d:\n", i + 1);
        inputComplexItem(&items[i]);
    }
    // Display the stored data for each item
    for (int i = 0; i < MAX_ITEMS; i++) {
        displayComplexItem(items[i]);
    }
    return 0;
}
```

Question 5: Write a C program that defines a structure to store information about books. Use a nested structure to store the author's details and a union to store either the number of pages or the publication year. Use typedef to simplify the structure and implement functions to input and display the information.

Sol:

```
#include <stdio.h>
#include <string.h>
// Define a structure to store author's details using typedef
typedef struct {
    char name[50];
```

```
    char country[50];
} Author;
// Define a union to store either the number of pages or the publication year using typedef
typedef union {
    int numPages;       // Number of pages
    int pubYear;        // Publication year
} BookInfo;
// Define a structure to store book information using typedef
typedef struct {
    char title[100];    // Book title
    Author author;      // Nested structure for author's details
    BookInfo info;      // Union to store number of pages or publication year
    int isPages;        // Flag to check whether the stored data is the number of pages or publication
year
} Book;
// Function to input book information
void inputBookInfo(Book *book) {
    printf("Enter book title: ");
    getchar();  // To consume the newline character left by the previous input
    fgets(book->title, sizeof(book->title), stdin);
    book->title[strcspn(book->title, "\n")] = '\0'; // Remove trailing newline
        // Input author's details
    printf("Enter author's name: ");
    fgets(book->author.name, sizeof(book->author.name), stdin);
```

```
    book->author.name[strcspn(book->author.name, "\n")] = '\0'; // Remove trailing newline
    printf("Enter author's country: ");
    fgets(book->author.country, sizeof(book->author.country), stdin);
    book->author.country[strcspn(book->author.country, "\n")] = '\0'; // Remove trailing newline
    // Choose whether to store the number of pages or publication year
    printf("Enter 1 to store number of pages, 2 to store publication year: ");
    scanf("%d", &book->isPages);
    if (book->isPages == 1) {
        printf("Enter number of pages: ");
        scanf("%d", &book->info.numPages);
    } else if (book->isPages == 2) {
        printf("Enter publication year: ");
        scanf("%d", &book->info.pubYear);
    } else {
        printf("Invalid choice! Please enter 1 or 2.\n");
    }
}
// Function to display book information
void displayBookInfo(Book book) {
    printf("\nBook Title: %s\n", book.title);
    printf("Author: %s\n", book.author.name);
    printf("Country: %s\n", book.author.country);
        if (book.isPages == 1) {
```

```
        printf("Number of Pages: %d\n", book.info.numPages);
    } else if (book.isPages == 2) {
        printf("Publication Year: %d\n", book.info.pubYear);
    } else {
        printf("Invalid data!\n");
    }
}
int main() {
    Book books[3];  // Array to store multiple books
    // Input and display information for each book
    for (int i = 0; i < 3; i++) {
        printf("\nEnter information for book %d:\n", i + 1);
        inputBookInfo(&books[i]);
    }
    // Display the information for each book
    for (int i = 0; i < 3; i++) {
        displayBookInfo(books[i]);
    }
    return 0;
}
```

Stacks Using Arrays and Linked List

1. **Question 1:** Write a C program to implement a stack using arrays. The program should include functions for all stack operations: push, pop, peek, isEmpty, and isFull. Demonstrate the working of the stack with sample data.

Sol: #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. **Question 2:** Develop a program to implement a stack using a linked list. Include functions for all stack operations: push, pop, peek, isEmpty, and isFull. Ensure proper memory management by handling dynamic allocation and deallocation.

Sol:

```
#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. **Question 3:** Create a C program to implement a stack using arrays. Include an additional operation to reverse the contents of the stack. Demonstrate the reversal operation with sample data.

Sol:

```
#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. **Question 4:** Write a program to implement a stack using a linked list. Extend the program to include an operation to merge two stacks. Demonstrate the merging operation by combining two stacks and displaying the resulting stack.

Sol:

```
#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. **Question 5:** Develop a program that implements a stack using arrays. Add functionality to check for balanced parentheses in an expression using the stack. Demonstrate this with sample expressions.

Sol:
```
#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. **Question 6:** Create a C program to implement a stack using a linked list. Extend the program to implement a stack-based evaluation of postfix expressions. Include all necessary stack operations and demonstrate the evaluation with sample expressions.

Sol:

```
#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;
}
```

1. **Student Admission Queue:** Write a program to simulate a student admission process. Implement a queue using arrays to manage students waiting for admission. Include operations to enqueue (add a student), dequeue (admit a student), and display the current queue of students.

Sol:

```
# 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]);
        }
    }
}
```

Library Book Borrowing Queue: Develop a program that simulates a library's book borrowing system. Use a queue to manage students waiting to borrow books. Include functions to add a student to the queue, remove a student after borrowing a book, and display the queue status.

Sol:

```
#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]);
            }
        }
    }
```

Cafeteria Token System: Create a program that simulates a cafeteria token system for students. Implement a queue using arrays to manage students waiting for their turn. Provide operations to issue tokens (enqueue), serve students (dequeue), and display the queue of students.

Sol:

```
#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]);
        }
    }
}
```

Classroom Help Desk Queue: Write a program to manage a help desk queue in a classroom. Use a queue to track students waiting for assistance. Include functions to add students to the queue, remove them once helped, and view the current queue.

Sol:

```
#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]);
        }
    }
}
```

Exam Registration Queue: Develop a program to simulate the exam registration process. Use a queue to manage the order of student registrations. Implement operations to add students to the queue, process their registration, and display the queue status.

Sol:

```
#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]);
        }
```

```
    }
}
```

School Bus Boarding Queue: Create a program that simulates the boarding process of a school bus. Implement a queue to manage the order in which students board the bus. Include functions to enqueue students as they arrive and dequeue them as they board.

Sol:

```
#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]);
        }
    }
}
```

Counseling Session Queue: Write a program to manage a queue for students waiting for a counseling session. Use an array-based queue to keep track of the students, with operations to add (enqueue) and serve (dequeue) students, and display the queue.

Sol:

```
#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]);
        }
    }
```

}

Sports Event Registration Queue: Develop a program that manages the registration queue for a school sports event. Use a queue to handle the order of student registrations, with functions to add, process, and display the queue of registered students.

Sol:

```
#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]);
        }
```

```
    }
}
```

Laboratory Equipment Checkout Queue: Create a program to simulate a queue for students waiting to check out laboratory equipment. Implement operations to add students to the queue, remove them once they receive equipment, and view the current queue.

Sol:

```
#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]);
```

```
        }
    }
}
```

Parent-Teacher Meeting Queue: Write a program to manage a queue for a parent-teacher meeting. Use a queue to organize the order in which parents meet the teacher. Include functions to enqueue parents, dequeue them after the meeting, and display the queue status.

Sol:

```
#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. Real-Time Sensor Data Processing:

Implement a queue using a linked list to store real-time data from various sensors (e.g., temperature, pressure). The system should enqueue sensor readings, process and dequeue the oldest data when a new reading arrives, and search for specific readings based on timestamps.

2. Task Scheduling in a Real-Time Operating System (RTOS):

Design a queue using a linked list to manage task scheduling in an RTOS. Each task should have a unique identifier, priority level, and execution time. Implement enqueue to add tasks, dequeue to remove the next task for execution, and search to find tasks by priority.

3. Interrupt Handling Mechanism:

Create a queue using a linked list to manage interrupt requests (IRQs) in an embedded system. Each interrupt should have a priority level and a handler function. Implement operations to enqueue new interrupts, dequeue the highest-priority interrupt, and search for interrupts by their source.

4. Message Passing in Embedded Communication Systems:

Implement a message queue using a linked list to handle inter-process communication in embedded systems. Each message should include a sender ID, receiver ID, and payload. Enqueue messages as they arrive, dequeue messages for processing, and search for messages from a specific sender.

5. Data Logging System for Embedded Devices:

Design a queue using a linked list to log data in an embedded system. Each log entry should contain a timestamp, event type, and description. Implement enqueue to add new logs, dequeue old logs when memory is low, and search for logs by event type.

6. Network Packet Management:

Create a queue using a linked list to manage network packets in an embedded router. Each packet should have a source IP, destination IP, and payload. Implement enqueue for incoming packets, dequeue for packets ready for transmission, and search for packets by IP address.

7. Firmware Update Queue:

Implement a queue using a linked list to manage firmware updates in an embedded system. Each update should include a version number, release notes, and file path. Enqueue updates as they become available, dequeue them for installation, and search for updates by version number.

8. Power Management Events:

Design a queue using a linked list to handle power management events in an embedded device. Each event should have a type (e.g., power on, sleep), timestamp, and associated action. Implement operations to enqueue events, dequeue events as they are handled, and search for events by type.

9. Command Queue for Embedded Systems:

Create a command queue using a linked list to handle user or system commands. Each command should have an ID, type, and parameters. Implement enqueue for new commands, dequeue for commands ready for execution, and search for commands by type.

10. Audio Buffering in Embedded Audio Systems:

Implement a queue using a linked list to buffer audio samples in an embedded audio system. Each buffer entry should include a timestamp and audio data. Enqueue new audio samples, dequeue samples for playback, and search for samples by timestamp.

11. Event-Driven Programming in Embedded Systems:

Design a queue using a linked list to manage events in an event-driven embedded system. Each event should have an ID, type, and associated data. Implement enqueue for new events, dequeue for event handling, and search for events by type or ID.

12. Embedded GUI Event Queue:

Create a queue using a linked list to manage GUI events (e.g., button clicks, screen touches) in an embedded system. Each event should have an event type, coordinates, and timestamp. Implement enqueue for new GUI events, dequeue for event handling, and search for events by type.

13. Serial Communication Buffer:

Implement a queue using a linked list to buffer data in a serial communication system. Each buffer entry should include data and its length. Enqueue new data chunks, dequeue them for transmission, and search for specific data patterns.

14. CAN Bus Message Queue:

Design a queue using a linked list to manage CAN bus messages in an embedded automotive system. Each message should have an ID, data length, and payload. Implement enqueue for incoming messages, dequeue for processing, and search for messages by ID.

15. Queue Management for Machine Learning Inference:

Create a queue using a linked list to manage input data for machine learning inference in an embedded system. Each entry should contain input features and metadata. Enqueue new data, dequeue it for inference, and search for specific input data by metadata.

Each problem requires creating a queue with the following operations using a linked list:

- **enqueue**: Add new elements to the queue.
- **dequeue**: Remove and process elements from the queue.
- **search**: Find elements based on specific criteria.
- **display**: Show all elements in the queue.

Solutions:

//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;
}
```