Variable, Static, Const, and Switch Case

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.

```
#include <stdio.h>

void myfunction();

int main(){
   myfunction();
   myfunction();
   myfunction();

   return 0;
}
void myfunction(){
   const int x =10;
   static int y =0;

   y++;

   switch(x){
      case 1:
         printf("The constant variable : %d\n",x);
         break;
      case 2:
         printf("The constant variable : %d\n",x);
         break;
      case 3:
         printf("The constant variable : %d\n",x);
         break;
      default:
         printf("The constant variable : %d\n",x);
   }
   printf("The static  variable : %d\n",y);
```

}

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.

```
#include <stdio.h>

void myfunction();

int main(){
    myfunction();
    myfunction();
    myfunction();
    myfunction();
    myfunction();
    return 0;
}
void myfunction(){
    static int x =0;
    const int var =3;

    if(x>var){
        printf("Maximum limit reached \n");
        return;
    }
    x++;
    switch (x)
    {
    case 1:
        printf("First time called \n");
        break;

    case 2:
        printf("second time called \n");
        break;
```

```
    case 3:
        printf("third time called \n");
        break;
    default:
        printf("FUnction called \n");
        break;
    }
}
```

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.

```
#include <stdio.h>

void myfun(int choice);

int main(){
    int choice;
    while(1){
        printf("\n1. Addition \n");
        printf("2. subtraction \n");
        printf("3. Multiplication\n");
        printf("4. Exit\n");
        printf("Enter choice ");
        scanf("%d",&choice);

    if(choice ==4){
        printf("Exiting.....\n");
        break;

    }
    if(choice>=1 && choice <=3){
        myfun(choice);
    }
    else{
        printf("Invalid Input\n");
    }
```

```
    }

}
void myfun(int choice){

    static int arr[] = {2,8,5,6,9};
    const int size =5;

    int result =(choice == 1) ? 0 : 1;
    switch (choice)
    {
    case 1:

        for(int i=0;i<size;i++){
            result +=arr[i];
        }
        printf("The Addition of array is %d",result);
        break;
    case 2:

        for(int i=0;i<size;i++){
            result -= arr[i];
        }
       printf("The subtraction of array is %d",result);
        break;
    case 3:

        for(int i=0;i<size;i++){
            result *=arr[i];
        }
        printf("The multiplication of array is %d",result);


    default:
        printf("Invalid\n");
        break;
    }
}
```

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.

```
#include <stdio.h>


void myFunc();

int main(){
    myFunc();
    myFunc();
    myFunc();
    myFunc();
    return 0;
}

void myFunc(){

    static int staticVar = 0;
    const int THRESHOLD =3;
    staticVar++;
    switch (staticVar)
    {
    case 1:
        printf("First time executed \n");
        break;
    case 2:
        printf("Second time executed \n");
        break;
    case 3:
        printf("Third time executed \n");
        break;
    default:
        if(staticVar>THRESHOLD){
            printf("Limit reached \n");
```

```
        }
        else{
            printf("Invalid \n");
        }
        break;
    }
}
```

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.

```
#include <stdio.h>

void myfunction();

int main(){
    for(int i=0;i<10;i++){
        myfunction();
    }
    return 0;

}

void myfunction(){
    static int counter =0;
    const int limit =5;

    counter++;
    switch (counter)
    {
    case 1:
        printf("First time executed \n");
        break;
```

```
    case 2:
        printf("second time executed \n");
        break;
    case 3:
        printf("Third time executed \n");
        break;
    case 4:
        printf("fouth time executed \n");
        break;
    case 5:
        printf("fifth time executed \n");
        break;

    default:
    printf("Counter has been reset\n");
        counter =0;

    }
}
```

Looping Statements, Pointers, Const with Pointers, Functions

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.

```
#include <stdio.h>

#define SIZE 5

void initializeArray(int *arr);    // Function to initialize the array using a single pointer
void modifyArray(int *arr);        // Function to modify the array using a pointer

int main() {
    int arr[SIZE];

    initializeArray(arr);
```

```
    printf("After initialization:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    modifyArray(arr);

    printf("After modification:\n");
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}

void initializeArray(int *arr) {
    for (int i = 0; i < SIZE; i++) {
        printf("Enter element %d: ", i);
        scanf("%d", &arr[i]);
    }
}

void modifyArray(int *arr) {
    const int *ptr = arr;

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

        arr[i] *= 2;
    }
}
```

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.

```
#include <stdio.h>
#define ROW 3
#define COLUMN 3
```

```
void intializeMatrix(int matrix[ROW][COLUMN]);
void TransposeMatrix(int matrix[ROW][COLUMN], int rows, int cols); // Call by reference
void TransposeMatrixByValue(int matrix[ROW][COLUMN], int row, int cols);  // Call by value

int main() {
    int matrix[ROW][COLUMN];

    // Initialize matrix with user input
    intializeMatrix(matrix);

    // Display original matrix
    printf("The original Matrix:\n");
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COLUMN; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    // Transpose by reference
    TransposeMatrix(matrix, ROW, COLUMN);
    printf("Matrix After Transpose (Call by Reference):\n");
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COLUMN; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    // Transpose by value and display the result
    printf("Transposed Matrix (Call by Value):\n");
    TransposeMatrixByValue(matrix, ROW, COLUMN);

    return 0;
}

// Function to initialize the matrix with user input
void intializeMatrix(int matrix[ROW][COLUMN]) {
```

```
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COLUMN; j++) {
            printf("Enter the element at position (%d,%d): ", i, j);
            scanf("%d", &matrix[i][j]);
        }
    }
}

// Function to transpose the matrix (call by reference)
void TransposeMatrix(int matrix[ROW][COLUMN], int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = i + 1; j < cols; j++) {
            // Swap elements to transpose the matrix
            int temp = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = temp;
        }
    }
}

// Function to transpose the matrix (call by value)
void TransposeMatrixByValue(int matrix[ROW][COLUMN], int row, int cols) {
    int temp[ROW][COLUMN];

    for (int i = 0; i < row; i++) {
        for (int j = 0; j < cols; j++) {
            temp[j][i] = matrix[i][j];
        }
    }

    // Print transposed matrix (without modifying the original matrix)
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", temp[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.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

void initializeArray(int *arr);
void printArray(int *arr);

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

   if (arr ==NULL){
    printf("Memory Allocation falied\n");
    return 1;
   }

   initializeArray(arr);
   printArray(arr);
   free(arr);
   printf("\nMemory dellocated \n");
   return 0;

}
// initializing array
void initializeArray(int *arr){
    int i=0;
    // for(int i=0;i<SIZE;i++){
    //     printf("The element at %d : ",i);
    //     scanf("%d",&arr[i]);
    // }
    while(i<SIZE){
        printf("The element at %d : ",i);
        scanf("%d",&arr[i]);
        i++;

    }
```

```
}


void printArray(int *arr){
    printf("The array elements are: ");
    for(int i=0;i<SIZE;i++){
        printf(" %d",arr[i]);

    }
}
```

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.

```
#include <stdio.h>
#define SIZE 5

void printArray(const int *arr);
void swapByReference(int (*arr1)[SIZE], int (*arr2)[SIZE]);
void swapByValue(int arr1[], int arr2[]);

int main() {
    int arr1[SIZE] = {1, 2, 3, 4, 5};
    int arr2[SIZE] = {6, 7, 8, 9, 10};

    printf("Original Arrays: \n");
    printf("Array 1: ");
    printArray(arr1);
    printf("Array 2: ");
    printArray(arr2);

    swapByReference(&arr1, &arr2);
    printf("\nArrays after swap by reference: \n");
    printf("Array 1: ");
    printArray(arr1);
    printf("Array 2: ");
```

```
    printArray(arr2);

    swapByValue(arr1, arr2);
    printf("\nArrays after swap by value: \n");
    printf("Array 1: ");
    printArray(arr1);
    printf("Array 2: ");
    printArray(arr2);

    return 0;
}

// Function to print array using const pointer (no modification allowed)
void printArray(const int *arr) {
    for (int i = 0; i < SIZE; i++) {
        printf("%d  ", arr[i]);
    }
    printf("\n");
}

// Swap two arrays by reference (this will modify the original arrays)
void swapByReference(int (*arr1)[SIZE], int (*arr2)[SIZE]) {
    int temp[SIZE];

    // Copy arr1 to temp
    for (int i = 0; i < SIZE; i++) {
        temp[i] = (*arr1)[i];
    }

    // Copy arr2 to arr1
    for (int i = 0; i < SIZE; i++) {
        (*arr1)[i] = (*arr2)[i];
    }

    // Copy temp (which holds original arr1) to arr2
    for (int i = 0; i < SIZE; i++) {
        (*arr2)[i] = temp[i];
    }
}

// Swap two arrays by value (this won't affect the original arrays)
```

```
void swapByValue(int arr1[], int arr2[]) {
    int temp[SIZE];

    // Copy arr1 to temp
    for (int i = 0; i < SIZE; i++) {
        temp[i] = arr1[i];
    }

    // Copy arr2 to arr1
    for (int i = 0; i < SIZE; i++) {
        arr1[i] = arr2[i];
    }

    // Copy temp (which holds original arr1) to arr2
    for (int i = 0; i < SIZE; i++) {
        arr2[i] = temp[i];
    }
}
```

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.

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

void frequencyCounter(const char *str);

int main(){
    char str[100];

    printf("Enter a string:  ");
    scanf(" %[^\n]s",str);

    frequencyCounter(str);
}
void frequencyCounter(const char *str){
    int frequency[256] = {0};
```

```
    const char *ptr = str;

    do{
        frequency[(unsigned char)*ptr]++;
        ptr++;
    } while(*ptr!='\0');
    printf("Character frequencies:\n");
    for (int i = 0; i < 256; i++) {
        if (frequency[i] > 0) {
            printf("'%c' : %d times\n", i, frequency[i]);
        }
    }

}
```

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

```
#include <stdio.h>

typedef struct {
    char address[150];
    char city[100];
}Address;
typedef struct {
    char name[50];
    char department[100];
    Address add;
}Employee;

#define MAX_EMPLOYEE 2

void inputEmpDetailes(Employee *emp);
void display(Employee emp);

int main(){
```

```
    Employee employees[MAX_EMPLOYEE];

    for(int i=0;i<MAX_EMPLOYEE;i++){
        printf("Enter detailes of Employee %d: \n",i+1);
        inputEmpDetailes(&employees[i]);
    }
    printf("\nEmployee Information:  \n");
    for(int i=0;i<MAX_EMPLOYEE;i++){
        printf("\nDetailes of Employee %d: \n",i+1);
        display(employees[i]);
    }
}

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

    printf("Department: ");
    scanf(" %[^\n]s",emp->department);

    printf("Address: ");
    scanf(" %[^\n]s",emp->add.address);
    printf("City: ");
    scanf(" %[^\n]s",emp->add.city);
}
void display(Employee emp){
    printf("Name: %s",emp.name);
    printf("Department: %s", emp.department);
    printf("Address: %s", emp.add.address);
    printf("City: %s", emp.add.city);
}
```

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).

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

```
#include <string.h>

typedef union{
    int intData;
    float floatData;
    char strData[100];
}Data;

typedef struct{
    int arr;
    Data data;
}Info;

void display(Info info);

int main(){
    Info dataArray[3];
    // integer
    dataArray[0].arr =1;
    dataArray[0].data.intData= 29;
    //float
    dataArray[1].arr =2;
    dataArray[1].data.floatData = 3.14;

    // string
    dataArray[2].arr =3;
    strcpy(dataArray[2].data.strData,"To be or not to be");

    for(int i=0;i<3;i++){
        display(dataArray[i]);
    }
    return 0;
}

void display(Info info){
    if(info.arr ==1){
        printf("Integer Data: %d\n", info.data.intData);
    }else if (info.arr == 2) {
        printf("Float Data: %.2f\n", info.data.floatData);
    }else if (info.arr == 3) {
        printf("String Data: %s\n", info.data.strData);
```

```
    } else {
        printf("Unknown data type!\n");
    }
}
```

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.

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

#define LENGTH 100
#define MAX 5

typedef union {
    int length;
    char reverse[LENGTH];
} stringInfo;

typedef struct {
    char name[LENGTH];
    stringInfo info;
    int isLength;
} NameInfo;

void reverse(char *str, char *reversed) {
    int len = strlen(str);
    for (int i = 0; i < len; i++) {
        reversed[i] = str[len - i - 1];
    }
    reversed[len] = '\0';
}

void displayNameInfo(NameInfo nameInfo) {
```

```
    printf("\nName: %s\n", nameInfo.name);
    if (nameInfo.isLength) {
        printf("Length: %d\n", nameInfo.info.length);
    } else {
        printf("Reversed: %s\n", nameInfo.info.reverse);
    }
}

int main() {
    NameInfo arr[MAX] = {
        {"Alice", {.length = 0}, 0},
        {"Bob", {.length = 0}, 0},
        {"Charlie", {.length = 0}, 0},
        {"David", {.length = 0}, 0},
        {"Eve", {.length = 0}, 0}
    };

    for (int i = 0; i < MAX; i++) {
        printf("Enter name %d: ", i + 1);
        scanf(" %[^\n]s", arr[i].name);
        arr[i].info.length = strlen(arr[i].name);
        arr[i].isLength = 1;
    }

    for (int i = 0; i < MAX; i++) {
        displayNameInfo(arr[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.

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

```
#define SIZE 3

typedef struct {
    int intData;
    float floatData;
} Structure2;

typedef union {
    int simpleInt;
    Structure2 nestedStructure;
} MyUnion;

typedef struct {
    char name[50];
    MyUnion data;
    int isNested;
} Structure1;

void display(Structure1 s) {
    printf("Name: %s\n", s.name);

    if (s.isNested == 0) {
        printf("Simple Integer: %d\n", s.data.simpleInt);
    } else {
        printf("Nested Structure - Integer: %d\n", s.data.nestedStructure.intData);
        printf("Nested Structure - Float: %.2f\n", s.data.nestedStructure.floatData);
    }
}

int main() {
    Structure1 arr[SIZE];

    strcpy(arr[0].name, "First Entry");
    arr[0].data.simpleInt = 10;
    arr[0].isNested = 0;

    strcpy(arr[1].name, "Second Entry");
    arr[1].data.nestedStructure.intData = 20;
    arr[1].data.nestedStructure.floatData = 3.14;
    arr[1].isNested = 1;
```

```
    strcpy(arr[2].name, "Third Entry");
    arr[2].data.simpleInt = 50;
    arr[2].isNested = 0;

    for (int i = 0; i < SIZE; i++) {
        display(arr[i]);
        printf("\n");
    }

    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.

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

#define MAX_BOOKS 3

typedef union {
    int numPages;
    int pubYear;
} BookInfo;

typedef struct {
    char name[50];
    char nationality[50];
} Author;

typedef struct {
    char title[100];
```

```
    Author author;
    BookInfo info;
    int isPage; // 0 for pages, 1 for publication year
} Book;

void inputBookInfo(Book *book);
void displayBookInfo(Book book);

int main() {
    Book books[MAX_BOOKS];

    for (int i = 0; i < MAX_BOOKS; i++) {
        printf("Enter details for book %d:\n", i + 1);
        inputBookInfo(&books[i]);
        printf("\n");
    }

    printf("\nDisplaying Book Information:\n");
    for (int i = 0; i < MAX_BOOKS; i++) {
        displayBookInfo(books[i]);
        printf("\n");
    }

    return 0;
}

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

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

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

    printf("Enter 0 for number of pages or 1 for publication year: ");
    scanf("%d", &book->isPage);

    if (book->isPage == 0) {
        printf("Enter number of pages: ");
```

```
        scanf("%d", &book->info.numPages);
    } else {
        printf("Enter publication year: ");
        scanf("%d", &book->info.pubYear);
    }
}

void displayBookInfo(Book book) {
    printf("Title: %s\n", book.title);
    printf("Author: %s\n", book.author.name);
    printf("Author Nationality: %s\n", book.author.nationality);

    if (book.isPage == 0) {
        printf("Number of Pages: %d\n", book.info.numPages);
    } else {
        printf("Publication Year: %d\n", book.info.pubYear);
    }
}
```

Stacks Using Arrays and Linked List

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.

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

#define MAX 5

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

void create(Stack *stack);
int isFUll(Stack *stack);
int isEmpty(Stack *stack);
void push(Stack *stack,int val);
```

```
int pop(Stack *stack);
int peek(Stack *stack);
void display(Stack *stack);

int main(){
   Stack stack;

   create(&stack);


   push(&stack, 10);
   push(&stack, 20);
   push(&stack, 30);
   push(&stack, 40);
   push(&stack, 50);
   push(&stack, 60);

   display(&stack);
   printf("Peek: %d\n", peek(&stack));

   printf("Popped: %d\n", pop(&stack));
   printf("Popped: %d\n", pop(&stack));

   display(&stack);

   return 0;
}

void create(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 val){
   if(isFUll(stack)){
      printf("Stack Overflow");
   }
```

```
    else{
        stack->arr[stack->top++] = val;
        printf("Pushed %d into the stack\n", val);
    }
}
int pop(Stack *stack){
    if(isEmpty(stack)){
        printf("Stack is empty, cannot pop\n");
        return -1;
    }else{
        return stack->arr[stack->top--];
    }
}
int  peek(Stack *stack){
    if(isEmpty(stack)){
        printf("Stack is empty\n");
        return -1;
    }else{
        return stack->arr[stack->top];
    }
}
void display(Stack *stack){
    if(isEmpty(stack)){
        printf("Stack is empty\n");
    }else{
        printf("Stack elements : ");
        for(int i=0;i<=stack->top;i++){
            printf("%d =>",stack->arr[i]);
        }
        printf("\n");
    }
}
```

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.

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

typedef struct Node {
```

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

typedef struct Stack {
    Node* top;
} Stack;

void create(Stack* stack);
int isEmpty(Stack* stack);
int isFull();
void push(Stack* stack, int value);
int pop(Stack* stack);
int peek(Stack* stack);
void display(Stack* stack);

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

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    push(&stack, 40);
    push(&stack, 50);

    display(&stack);
    printf("Peek: %d\n", peek(&stack));

    printf("Popped: %d\n", pop(&stack));
    printf("Popped: %d\n", pop(&stack));

    display(&stack);

    return 0;
}

void create(Stack* stack) {
    stack->top = NULL;
}
```

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

int isFull() {
    Node* temp = (Node*)malloc(sizeof(Node));
    if (temp == NULL) {
        return 1;
    }
    free(temp);
    return 0;
}

void push(Stack* stack, int value) {
    if (isFull()) {
        printf("Stack Overflow: Unable to allocate memory\n");
        return;
    }
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = stack->top;
    stack->top = newNode;
    printf("Pushed %d into the stack\n", value);
}

int pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow: Stack is empty, cannot pop\n");
        return -1;
    }
    Node* temp = stack->top;
    int poppedValue = temp->data;
    stack->top = temp->next;
    free(temp);
    return poppedValue;
}

int peek(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return -1;
```

```
    }
    return stack->top->data;
}

void display(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
    } else {
        Node* current = stack->top;
        printf("Stack elements: ");
        while (current != NULL) {
            printf("%d => ", current->data);
            current = current->next;
        }
        printf("\n");
    }
}
```

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.

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

#define MAX 5

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

void create(Stack* stack);
int isFull(Stack* stack);
int isEmpty(Stack* stack);
void push(Stack* stack, int val);
int pop(Stack* stack);
int peek(Stack* stack);
void display(Stack* stack);
void reverse(Stack* stack);
```

```
int main() {
    Stack stack;
    create(&stack);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    push(&stack, 40);
    push(&stack, 50);

    printf("Original stack:\n");
    display(&stack);

    reverse(&stack);

    printf("Reversed stack:\n");
    display(&stack);

    return 0;
}

void create(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 val) {
    if (isFull(stack)) {
        printf("Stack Overflow\n");
    } else {
        stack->arr[++stack->top] = val;
        printf("Pushed %d into the stack\n", val);
    }
```

```
}

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

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

void display(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements: ");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d ", stack->arr[i]);
        }
        printf("\n");
    }
}

void reverse(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty, nothing to reverse\n");
        return;
    }
    int tempStack[MAX];
    int index = 0;

    while (!isEmpty(stack)) {
```

```
        tempStack[index++] = pop(stack);
    }

    for (int i = 0; i < index; i++) {
        push(stack, tempStack[i]);
    }
}
```

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.

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

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

typedef struct Stack {
    Node* top;
} Stack;

void create(Stack* stack);
int isEmpty(Stack* stack);
void push(Stack* stack, int val);
int pop(Stack* stack);
void display(Stack* stack);
void mergeStacks(Stack* stack1, Stack* stack2, Stack* result);

int main() {
    Stack stack1, stack2, mergedStack;

    create(&stack1);
    create(&stack2);
    create(&mergedStack);
```

```
    push(&stack1, 10);
    push(&stack1, 20);
    push(&stack1, 30);

    push(&stack2, 40);
    push(&stack2, 50);
    push(&stack2, 60);

    printf("Stack 1:\n");
    display(&stack1);

    printf("Stack 2:\n");
    display(&stack2);

    mergeStacks(&stack1, &stack2, &mergedStack);

    printf("Merged Stack:\n");
    display(&mergedStack);

    return 0;
}

void create(Stack* stack) {
    stack->top = NULL;
}

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

void push(Stack* stack, int val) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        return;
    }
    newNode->data = val;
    newNode->next = stack->top;
    stack->top = newNode;
    printf("Pushed %d into the stack\n", val);
}
```

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

void display(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
    } else {
        Node* current = stack->top;
        printf("Stack elements: ");
        while (current) {
            printf("%d ", current->data);
            current = current->next;
        }
        printf("\n");
    }
}

void mergeStacks(Stack* stack1, Stack* stack2, Stack* result) {
    Node* tempStack1 = stack1->top;
    Node* tempStack2 = stack2->top;

    while (tempStack1) {
        push(result, tempStack1->data);
        tempStack1 = tempStack1->next;
    }

    while (tempStack2) {
        push(result, tempStack2->data);
        tempStack2 = tempStack2->next;
    }
}
```

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.

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

#define MAX 100

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

void create(Stack* stack);
int isFull(Stack* stack);
int isEmpty(Stack* stack);
void push(Stack* stack, char val);
char pop(Stack* stack);
char peek(Stack* stack);
int checkBalancedParentheses(const char* expression);

int main() {
    char expression[MAX];

    printf("Enter an expression: ");
    scanf(" %[^\n]s", expression);

    if (checkBalancedParentheses(expression)) {
        printf("The parentheses in the expression are balanced.\n");
    } else {
        printf("The parentheses in the expression are not balanced.\n");
    }

    return 0;
}

void create(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 val) {
    if (isFull(stack)) {
        printf("Stack Overflow\n");
        return;
    }
    stack->arr[++stack->top] = val;
}

char pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow\n");
        return '\0';
    }
    return stack->arr[stack->top--];
}

char peek(Stack* stack) {
    if (isEmpty(stack)) {
        return '\0';
    }
    return stack->arr[stack->top];
}

int checkBalancedParentheses(const char* expression) {
    Stack stack;
    create(&stack);

    for (int i = 0; i < strlen(expression); i++) {
        char ch = expression[i];
```

```
        if (ch == '(' || ch == '{' || ch == '[') {
            push(&stack, ch);
        } else if (ch == ')' || ch == '}' || ch == ']') {
            if (isEmpty(&stack)) {
                return 0;
            }

            char topChar = pop(&stack);

            if ((ch == ')' && topChar != '(') ||
                (ch == '}' && topChar != '{') ||
                (ch == ']' && topChar != '[')) {
                return 0;
            }
        }
    }

    return isEmpty(&stack);
}
```

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.

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

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

typedef struct {
    Node* top;
} Stack;
```

```
void create(Stack* stack);
int isEmpty(Stack* stack);
void push(Stack* stack, int value);
int pop(Stack* stack);
int peek(Stack* stack);
int evaluatePostfix(const char* expression);

int main() {
    char expression[100];

    printf("Enter a postfix expression: ");
    scanf(" %[^\n]s", expression);

    int result = evaluatePostfix(expression);
    printf("The result of the postfix expression is: %d\n", result);

    return 0;
}

void create(Stack* stack) {
    stack->top = NULL;
}

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

void push(Stack* stack, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = stack->top;
    stack->top = newNode;
}

int pop(Stack* stack) {
    if (isEmpty(stack)) {
```

```
        printf("Stack Underflow\n");
        exit(1);
    }
    Node* temp = stack->top;
    int value = temp->data;
    stack->top = stack->top->next;
    free(temp);
    return value;
}

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

int evaluatePostfix(const char* expression) {
    Stack stack;
    create(&stack);

    for (int i = 0; expression[i] != '\0'; i++) {
        char ch = expression[i];

        if (isspace(ch)) {
            continue;
        }

        if (isdigit(ch)) {
            push(&stack, ch - '0');
        } else {
            int operand2 = pop(&stack);
            int operand1 = pop(&stack);

            switch (ch) {
                case '+':
                    push(&stack, operand1 + operand2);
                    break;
                case '-':
                    push(&stack, operand1 - operand2);
```

```
                break;
            case '*':
                push(&stack, operand1 * operand2);
                break;
            case '/':
                if (operand2 == 0) {
                    printf("Division by zero error\n");
                    exit(1);
                }
                push(&stack, operand1 / operand2);
                break;
            default:
                printf("Invalid operator: %c\n", ch);
                exit(1);
        }
    }
}

    return pop(&stack);
}


#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);
    enqueue(&q,10);
    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 \n");
    }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("Q is empty");
    }
```

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

implementation thrugh link list

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

struct Node{
    int data;
    struct Node *next;
} &front = NULL, *rear =NULL;

void enqueue(int);
int dequeue();

int main(){
    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);

    return 0;
}

void enqueue(int){
    struct Node *t;
    t = (struct Node *)malloc(sizeof(struct Node));
    if(t == NULL){
        printf("Queue is full \n");
    }else{
        t->data = x;

    }
}
```

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

struct Node {
    int data;
    struct Node *next;
} *front = NULL, *rear = NULL;

void enqueue(int);
int dequeue();
void display();

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);
    enqueue(50);
    display();
    printf("%d \n", dequeue());
    display();
    return 0;
}

void enqueue(int x) {
    struct Node *t;
    t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Queue is full \n");
    } else {
        t->data = x;
        t->next = NULL;
```

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

void display() {
    struct Node *p = front;
    if (p == NULL) {
        printf("Queue is empty\n");
        return;
    }
    while (p) {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("\n");
}

int dequeue() {
    int x = -1;
    struct Node *t;
    if (front == NULL) {
        printf("Queue is already empty \n");
    } else {
        x = front->data;
        t = front;
        front = front->next;
        free(t);
        if (front == NULL) {
            rear = NULL;  // If queue becomes empty, reset rear pointer.
        }
    }
    return x;
}
```

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.

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

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

void create(struct Queue *,int);
void enqueue(struct Queue *,char data[]);
void dequeue(struct Queue *);
void display(struct Queue *q);


int main(){
    struct Queue q;
    create(&q,5);
    enqueue(&q, "Bhumika");
    enqueue(&q, "venkat");
    enqueue(&q, "Marie");
    display(&q);
    dequeue(&q);
    display(&q);
}

void create(struct Queue *q,int size){
    q->size = size;
    q->front = q->rear = -1;
    q->Q = (char **)malloc (q->size * sizeof(char *));
}
void enqueue(struct Queue *q,char data[]){
    if(q->rear == q->size-1){
```

```
        printf("Admissions over \n");
    }else{
        q->rear++;
        q->Q[q->rear] = (char *) malloc ((strlen(data)+1) * sizeof(char));
        strcpy(q->Q[q->rear], data);
    }
}
void dequeue(struct Queue *q){
    if(q->front == q->rear){
        printf("No students in queue \n");
    }else{
        q->front++;
        printf("Admitted Students: %s \n",q->Q[q->front]);
        free(q->Q[q->front]);
    }

}

void display(struct Queue *q){
    if(q->front == q->rear){
        printf("No students in queue \n");
    }
    else{
        printf("Current Queue students: ");
        for(int i = q->front +1;i<= q->rear;i++){
            printf("%s\t",q->Q[i]);
        }
        printf("\n");
    }
}
```

2) 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.

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

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

void create(struct Queue *q, int size);
void enqueue(struct Queue *q, char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

    enqueue(&q, "Bhumika");
    enqueue(&q, "Venkat");
    enqueue(&q, "Marie");

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

    return 0;
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->Q = (char **)malloc(q->size * sizeof(char *));
    if (q->Q == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}
```

```
void enqueue(struct Queue *q, char data[]) {
    if (q->rear == q->size - 1) {
        printf("No space for more students. Queue is full.\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc((strlen(data) + 1) * sizeof(char));  // Allocate
memory for student name
        if (q->Q[q->rear] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
        strcpy(q->Q[q->rear], data);
    }
}

void dequeue(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        q->front++;
        printf("Student borrowed a book: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}

void display(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        printf("Students waiting for book borrowing: ");
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("%s\t", q->Q[i]);
        }
        printf("\n");
    }
}
```

3) 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.

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

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

void create(struct Queue *q,int size);
void enqueue(struct Queue *q,char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

    enqueue(&q, "Bhumika");
    enqueue(&q, "Venkat");
    enqueue(&q, "Marie");

    printf("Current queue of students: \n");
    display(&q);

    printf("\nServing Student.....\n");
    dequeue(&q);

    printf("\nQueue after serving one student: \n");
    display(&q);

    return 0;
}
```

```
void create(struct Queue *q,int size){
    q->size =size;
    q->front = q->rear =-1;
    q->Q = (char **) malloc(q->size * sizeof(char *));
}
void enqueue(struct Queue *q,char data[]){
     if (q->rear == q->size - 1) {
        printf("Queue is full. No more students can join.\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc((strlen(data) + 1) * sizeof(char));
        if (q->Q[q->rear] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
        strcpy(q->Q[q->rear], data);
    }

}
void dequeue(struct Queue *q){
    if(q->front== q->rear){
        printf("No students in the queue.\n");
    }else{
        q->front++;
        printf("Student served: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}
void display(struct Queue *q){
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        printf("Students waiting for their turn: ");
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("%s\t", q->Q[i]);
        }
        printf("\n");
    }
}
```

4) 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.

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

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

void create(struct Queue *q, int size);
void enqueue(struct Queue *q, char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

    enqueue(&q, "Bhumika");
    enqueue(&q, "Venkat");
    enqueue(&q, "Marie");

    printf("Current queue of students waiting for assistance: \n");
    display(&q);

    printf("\nHelping student...\n");
    dequeue(&q);

    printf("\nQueue after helping one student: \n");
```

```
    display(&q);

    return 0;
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->Q = (char **)malloc(q->size * sizeof(char *));
    if (q->Q == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}

void enqueue(struct Queue *q, char data[]) {
    if (q->rear == q->size - 1) {
        printf("Queue is full. No more students can join.\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc((strlen(data) + 1) * sizeof(char));
        if (q->Q[q->rear] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
        strcpy(q->Q[q->rear], data);
    }
}

void dequeue(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        q->front++;
        printf("Student helped: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}

void display(struct Queue *q) {
    if (q->front == q->rear) {
```

```
        printf("No students in the queue.\n");
    } else {
        printf("Students waiting for help: ");
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("%s\t", q->Q[i]);
        }
        printf("\n");
    }
}
```

5) 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.

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

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

void create(struct Queue *q, int size);
void enqueue(struct Queue *q, char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

```
    enqueue(&q, "Bhumika");
    enqueue(&q, "Venkat");
    enqueue(&q, "Marie");

    printf("Current exam registration queue: \n");
    display(&q);

    printf("\nProcessing registration...\n");
    dequeue(&q);

    printf("\nQueue after processing one registration: \n");
    display(&q);

    return 0;
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->Q = (char **)malloc(q->size * sizeof(char *));
    if (q->Q == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}

void enqueue(struct Queue *q, char data[]) {
    if (q->rear == q->size - 1) {
        printf("Queue is full. No more students can register.\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc((strlen(data) + 1) * sizeof(char));
        if (q->Q[q->rear] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
        strcpy(q->Q[q->rear], data);
    }
}

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

```
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        q->front++;
        printf("Student registered: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}

void display(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        printf("Students waiting to register: ");
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("%s\t", q->Q[i]);
        }
        printf("\n");
    }
}
```

6) 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.

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

struct Queue {
    int size;
    int front;
```

```
    int rear;
    char **Q;
};

void create(struct Queue *q, int size);
void enqueue(struct Queue *q, char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

    enqueue(&q, "Bhumika");
    enqueue(&q, "Venkat");
    enqueue(&q, "Marie");

    printf("Students waiting to board the bus: \n");
    display(&q);

    printf("\nBoarding student...\n");
    dequeue(&q);

    printf("\nQueue after boarding one student: \n");
    display(&q);

    return 0;
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->Q = (char **)malloc(q->size * sizeof(char *));
    if (q->Q == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}

void enqueue(struct Queue *q, char data[]) {
    if (q->rear == q->size - 1) {
```

```
        printf("Queue is full. No more students can board the bus.\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc((strlen(data) + 1) * sizeof(char));
        if (q->Q[q->rear] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
        strcpy(q->Q[q->rear], data);
    }
}

void dequeue(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        q->front++;
        printf("Student boarded the bus: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}

void display(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        printf("Students waiting to board the bus: ");
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("%s\t", q->Q[i]);
        }
        printf("\n");
    }
}
```

7) 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.

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

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

void create(struct Queue *q, int size);
void enqueue(struct Queue *q, char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

    enqueue(&q, "Bhumika");
    enqueue(&q, "Venkat");
    enqueue(&q, "Marie");

    printf("Students waiting for counseling: \n");
    display(&q);

    printf("\nCounseling student...\n");
    dequeue(&q);

    printf("\nQueue after counseling one student: \n");
    display(&q);

    return 0;
```

```
}

void create(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->Q = (char **)malloc(q->size * sizeof(char *));
    if (q->Q == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
}

void enqueue(struct Queue *q, char data[]) {
    if (q->rear == q->size - 1) {
        printf("Queue is full. No more students can wait for counseling.\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc((strlen(data) + 1) * sizeof(char));
        if (q->Q[q->rear] == NULL) {
            printf("Memory allocation failed!\n");
            exit(1);
        }
        strcpy(q->Q[q->rear], data);
    }
}

void dequeue(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        q->front++;
        printf("Student served for counseling: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}

void display(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        printf("Students waiting for counseling: ");
```

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

8) 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.

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

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

void create(struct Queue *q, int size);
void enqueue(struct Queue *q, char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

```
    create(&q, 5);

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

    printf("Sports Event Registration Queue:\n");
    display(&q);

    dequeue(&q);
    printf("\nQueue after processing one registration:\n");
    display(&q);

    return 0;
}

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

void enqueue(struct Queue *q, char data[]) {
    if (q->rear == q->size - 1) {
        printf("Queue is full. No more registrations can be accepted.\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc((strlen(data) + 1) * sizeof(char));
        strcpy(q->Q[q->rear], data);
    }
}

void dequeue(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        q->front++;
        printf("Processed registration for: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}
```

```
void display(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("%s\t", q->Q[i]);
        }
        printf("\n");
    }
}
```

9) 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.

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

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

```
void create(struct Queue *q, int size);
void enqueue(struct Queue *q, char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

    enqueue(&q, "Student1");
    enqueue(&q, "Student2");
    enqueue(&q, "Student3");

    printf("Lab Equipment Checkout Queue:\n");
    display(&q);

    dequeue(&q);
    printf("\nQueue after serving one student:\n");
    display(&q);

    return 0;
}

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

void enqueue(struct Queue *q, char data[]) {
    if (q->rear == q->size - 1) {
        printf("Queue is full. No more students can be added.\n");
    } else {
        q->rear++;
        q->Q[q->rear] = (char *)malloc((strlen(data) + 1) * sizeof(char));
        strcpy(q->Q[q->rear], data);
    }
}

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

```
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        q->front++;
        printf("Equipment issued to: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}

void display(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No students in the queue.\n");
    } else {
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("%s\t", q->Q[i]);
        }
        printf("\n");
    }
}
```

10) 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.

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

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

void create(struct Queue *q, int size);
void enqueue(struct Queue *q, char data[]);
void dequeue(struct Queue *q);
void display(struct Queue *q);

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

    enqueue(&q, "Parent1");
    enqueue(&q, "Parent2");
    enqueue(&q, "Parent3");

    printf("Parent-Teacher Meeting Queue:\n");
    display(&q);

    dequeue(&q);
    printf("\nQueue after one meeting:\n");
    display(&q);

    return 0;
}

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

void enqueue(struct Queue *q, char data[]) {
    if (q->rear == q->size - 1) {
        printf("Queue is full. No more parents can be added.\n");
    } else {
        q->rear++;
```

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

void dequeue(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No parents in the queue.\n");
    } else {
        q->front++;
        printf("Meeting completed for: %s\n", q->Q[q->front]);
        free(q->Q[q->front]);
    }
}

void display(struct Queue *q) {
    if (q->front == q->rear) {
        printf("No parents in the queue.\n");
    } else {
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("%s\t", q->Q[i]);
        }
        printf("\n");
    }
}
```