QUEST VIVA PROGRAMS
==========
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.

```
#include <stdio.h>

void testFunction() {
    // Declare a static variable
    static int count = 0;

    // Declare a const variable
    const int caseType = 2; // You can change this to 1 or 3 to test different cases

    // Increment the static variable each time the function is called
    count++;

    // Print the static variable to show persistence across multiple calls
    printf("Function called %d time(s). Static variable 'count' = %d\n", count, count);

    // Switch case to check the value of the const variable
    switch (caseType) {
        case 1:
            printf("Case 1: The value of the const variable is 1.\n");
            break;
        case 2:
            printf("Case 2: The value of the const variable is 2.\n");
            break;
        case 3:
            printf("Case 3: The value of the const variable is 3.\n");
            break;
        default:
            printf("Default case: Invalid value for const variable.\n");
            break;
    }
}

int main() {
    // Call the function three times
    testFunction();
    printf("\n");
    testFunction();
    printf("\n");
    testFunction();

    return 0;
}
```

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.

```
#include <stdio.h>

#define MAX_CALLS 3 // Constant to define the maximum number of calls

void trackCalls(void);

int main() {
    // Call the trackCalls function multiple times
    trackCalls(); // First call
    trackCalls(); // Second call
    trackCalls(); // Third call
    trackCalls(); // Attempt to exceed the limit

    return 0;
}

void trackCalls(void) {
    static int callCount = 0; // Static variable to track number of calls

    // Check if the function has exceeded the maximum allowed calls
    if (callCount >= MAX_CALLS) {
        printf("Function call limit reached. Further calls are not allowed.\n");
        return;
    }

    callCount++; // Increment the call count

    // Use a switch-case to print different messages based on the call count
    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.

```
#include <stdio.h>

#define ARRAY_SIZE 5  // Define the size of the array

// Function to perform operations on the static array
void manipulateArray(int operation) {
    static int arr[ARRAY_SIZE] = {0};  // Static array to store values across function calls
```

```
    int index, value;

    switch (operation) {
        case 1:  // Add a value to the array
            printf("Enter the index (0 to %d) to add a value: ", ARRAY_SIZE - 1);
            scanf("%d", &index);
            if (index >= 0 && index < ARRAY_SIZE) {
                printf("Enter the value to add: ");
                scanf("%d", &value);
                arr[index] += value;  // Add value to the element at the specified index
                printf("Value at index %d updated to: %d\n", index, arr[index]);
            } else {
                printf("Invalid index!\n");
            }
            break;

        case 2:  // Subtract a value from the array
            printf("Enter the index (0 to %d) to subtract a value: ", ARRAY_SIZE - 1);
            scanf("%d", &index);
            if (index >= 0 && index < ARRAY_SIZE) {
                printf("Enter the value to subtract: ");
                scanf("%d", &value);
                arr[index] -= value;  // Subtract value from the element at the specified index
                printf("Value at index %d updated to: %d\n", index, arr[index]);
            } else {
                printf("Invalid index!\n");
            }
            break;

        case 3:  // Multiply a value in the array
            printf("Enter the index (0 to %d) to multiply a value: ", ARRAY_SIZE - 1);
            scanf("%d", &index);
            if (index >= 0 && index < ARRAY_SIZE) {
                printf("Enter the value to multiply: ");
                scanf("%d", &value);
                arr[index] *= value;  // Multiply the element at the specified index
                printf("Value at index %d updated to: %d\n", index, arr[index]);
            } else {
                printf("Invalid index!\n");
            }
            break;

        case 4:  // Display the current values in the array
            printf("Current array values: ");
            for (int i = 0; i < ARRAY_SIZE; i++) {
                printf("%d ", arr[i]);
            }
            printf("\n");
            break;

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

```
int main() {
    int choice;

    do {
        printf("\nChoose an operation:\n");
        printf("1. Add to an element\n");
        printf("2. Subtract from an element\n");
        printf("3. Multiply an element\n");
        printf("4. Display array values\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice != 0) {
            manipulateArray(choice);  // Call the function to manipulate the array
        }
    } while (choice != 0);

    printf("Exiting the program.\n");

    return 0;
}
```

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

```
#include <stdio.h>

int main() {
    // Define a constant threshold value
    const int THRESHOLD = 5;

    // Static variable to keep track of the count of case executions
    static int executionCount = 0;

    // Variable to simulate user input for switch case execution
    int action;

    // Loop to simulate repeated actions
    for (int i = 0; i < 10; ++i) {
        printf("Enter a number between 1 and 3 to trigger a case (0 to exit): ");
        scanf("%d", &action);

        switch (action) {
            case 1:
                printf("Case 1 executed.\n");
                executionCount++; // Increment static counter
                break;
            case 2:
                printf("Case 2 executed.\n");
                executionCount++; // Increment static counter
                break;
```

```
            case 3:
                printf("Case 3 executed.\n");
                executionCount++; // Increment static counter
                break;
            case 0:
                printf("Exiting...\n");
                return 0; // Exit the program
            default:
                printf("Invalid input, please try again.\n");
        }

        // Check if the execution count has reached the threshold
        if (executionCount >= THRESHOLD) {
            printf("Threshold reached! Executing special action...\n");
            executionCount = 0; // Reset the counter after the threshold is reached
        }

        printf("Current execution count: %d\n", executionCount);
    }

    return 0;
}
```

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.

```
#include <stdio.h>

#define LIMIT 5 // Define the const limit

void counterFunction() {
    static int counter = 0; // Static counter to persist between function calls

    counter++; // Increment the counter

    // Use a switch-case to print different messages based on the value of the counter
    switch (counter) {
        case 1:
            printf("Counter reached 1\n");
            break;
        case 2:
            printf("Counter reached 2\n");
            break;
        case 3:
            printf("Counter reached 3\n");
            break;
        case 4:
            printf("Counter reached 4\n");
            break;
        case 5:
            printf("Counter reached 5\n");
            // Reset the counter after 5 calls
            counter = 0;
```

```
            break;
        default:
            break;
    }
}

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

    // Attempting to modify the const variable (this will cause a compilation error)
    // LIMIT = 10; // Uncommenting this line will cause a compilation error

    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

```
#include <stdio.h>

#define SIZE 5

// Function to initialize an array using a for loop
void initializeArray(int *arr) {
    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;  // Initialize elements to 1, 2, 3, ..., SIZE
    }
}

// Function to modify the array elements using a double pointer
void modifyArray(int **arr) {
    for (int i = 0; i < SIZE; i++) {
        (*arr)[i] *= 2;  // Double the value of each element
    }
}

// Function to print the array elements (to verify the changes)
void printArray(const int *arr) {
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[SIZE];  // Declare an array

    // Initialize the array
    initializeArray(arr);
```

```
    printf("Array after initialization:\n");
    printArray(arr);

    // Modify the array using a double pointer
    modifyArray(&arr);

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

    return 0;
}
```

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>
#include <stdlib.h>

void transposeByValue(int **matrix, int rows, int cols);
void transposeByReference(int ***matrix, int rows, int cols);
void printMatrix(int **matrix, int rows, int cols);

int main() {
    int rows, cols;

    // Reading matrix dimensions from the user
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    printf("Enter number of columns: ");
    scanf("%d", &cols);

    // Dynamically allocate memory for the matrix
    int **matrix = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        matrix[i] = (int *)malloc(cols * sizeof(int));
    }

    // Reading the matrix elements from the user
    printf("Enter the elements of the matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Print the original matrix
    printf("\nOriginal Matrix:\n");
    printMatrix(matrix, rows, cols);

    // Transpose by Value
    printf("\nTransposed Matrix (Call by Value):\n");
    transposeByValue(matrix, rows, cols);  // This will not modify the original matrix
```

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

    // Transpose by Reference
    printf("\nTransposed Matrix (Call by Reference):\n");
    transposeByReference(&matrix, rows, cols);  // This will modify the original matrix
    printMatrix(matrix, rows, cols);

    // Free dynamically allocated memory
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);

    return 0;
}

// Function to transpose the matrix using call by value
void transposeByValue(int **matrix, int rows, int cols) {
    // Creating a temporary matrix to store the transpose
    int **transposed = (int **)malloc(cols * sizeof(int *));
    for (int i = 0; i < cols; i++) {
        transposed[i] = (int *)malloc(rows * sizeof(int));
    }

    // Performing the transpose operation
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            transposed[j][i] = matrix[i][j];
        }
    }

    // Print the transposed matrix (not modifying the original matrix)
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            printf("%d ", transposed[i][j]);
        }
        printf("\n");
    }

    // Free the temporary transposed matrix memory
    for (int i = 0; i < cols; i++) {
        free(transposed[i]);
    }
    free(transposed);
}

// Function to transpose the matrix using call by reference
void transposeByReference(int ***matrix, int rows, int cols) {
    // Allocate memory for the transposed matrix
    int **transposed = (int **)malloc(cols * sizeof(int *));
    for (int i = 0; i < cols; i++) {
        transposed[i] = (int *)malloc(rows * sizeof(int));
    }

    // Perform the transpose operation
```

```
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            transposed[j][i] = (*matrix)[i][j];
        }
    }

    // Deallocate the original matrix and point to the transposed matrix
    for (int i = 0; i < rows; i++) {
        free((*matrix)[i]);
    }
    free(*matrix);

    // Assign the transposed matrix to the original matrix pointer
    *matrix = transposed;
}

// Function to print the matrix
void printMatrix(int **matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
```

3.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>

// Function to initialize the array using a while loop
void initializeArray(int *arr, int size) {
    int i = 0;
    while (i < size) {
        arr[i] = i + 1;  // Initialize the array with values 1, 2, 3, ...
        i++;
    }
}

// Function to print the array using a const pointer to prevent modification
void printArray(const int *arr, int size) {
    int i = 0;
    while (i < size) {
        printf("%d ", arr[i]);
        i++;
    }
    printf("\n");
}

int main() {
    int size;
```

```
    // Ask the user for the size of the array
    printf("Enter the size of the array: ");
    scanf("%d", &size);

    // Dynamically allocate memory for the array
    int *arr = (int *)malloc(size * sizeof(int));

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

    // Initialize the array
    initializeArray(arr, size);

    // Print the array
    printf("Array elements: ");
    printArray(arr, size);

    // Free the dynamically allocated memory
    free(arr);

    return 0;
}
```

4.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  // Size of the arrays

// Function to swap arrays using call by value (copying the array pointers)
void swapByValue(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;
    }
}

// Function to swap arrays using call by reference (passing array pointers directly)
void swapByReference(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;
    }
}

// Function to print the array
```

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

int main() {
    // Declare two arrays
    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, SIZE);
    printf("Array 2: ");
    printArray(arr2, SIZE);

    // Swap arrays using call by value
    swapByValue(arr1, arr2, SIZE);
    printf("\nArrays after swapping using call by value:\n");
    printf("Array 1: ");
    printArray(arr1, SIZE);
    printf("Array 2: ");
    printArray(arr2, SIZE);

    // Swap arrays back to original using call by reference
    swapByReference(&arr1, &arr2, SIZE);
    printf("\nArrays after swapping back using call by reference:\n");
    printf("Array 1: ");
    printArray(arr1, SIZE);
    printf("Array 2: ");
    printArray(arr2, SIZE);

    return 0;
}
```

5.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>

#define MAX_SIZE 100

// Function to read a string from the user
void readString(char *str) {
    printf("Enter a string: ");
    // Using scanf to read the string (up to MAX_SIZE - 1 characters)
    scanf("%99[^\n]", str);  // Read until newline is encountered, with a limit on input size
}

// Function to count the frequency of each character
```

```
void countFrequency(const char *str) {
    int frequency[256] = {0};  // Array to store frequency of each ASCII character

    // Using a do-while loop to traverse through the string
    int i = 0;
    do {
        frequency[(unsigned char)str[i]]++;  // Increment the frequency for each character
        i++;
    } while (str[i] != '\0');  // Continue until the end of the string

    // Display the frequency of each character
    printf("Character frequency:\n");
    for (int i = 0; i < 256; i++) {
        if (frequency[i] > 0) {
            printf("'%c' appears %d times\n", i, frequency[i]);
        }
    }
}

int main() {
    char str[MAX_SIZE];

    // Read a string from the user
    readString(str);

    // Count and display the frequency of each character in the string
    countFrequency(str);

    return 0;
}
```

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.

```
#include <stdio.h>

// Define the Address structure using typedef
typedef struct {
    char street[100];
    char city[50];
    char state[50];
    char zipCode[20];
} Address;

// Define the Employee structure using typedef
typedef struct {
    int id;
    char name[100];
    float salary;
    Address address;  // Nested structure
} Employee;
```

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

    printf("Enter employee name: ");
    // Read the name, assuming no spaces in the name
    scanf("%s", emp->name);

    printf("Enter employee salary: ");
    scanf("%f", &emp->salary);

    printf("Enter street address: ");
    scanf(" %[^\n]", emp->address.street);  // This allows spaces in the street address

    printf("Enter city: ");
    scanf(" %[^\n]", emp->address.city);    // This allows spaces in the city name

    printf("Enter state: ");
    scanf(" %[^\n]", emp->address.state);   // This allows spaces in the state name

    printf("Enter ZIP code: ");
    scanf("%s", emp->address.zipCode);      // ZIP code doesn't need spaces
}

void displayEmployeeData(Employee emp) {
    printf("\nEmployee ID: %d\n", emp.id);
    printf("Employee Name: %s\n", emp.name);
    printf("Employee Salary: %.2f\n", emp.salary);
    printf("Employee 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: %s\n", emp.address.zipCode);
}

int main() {
    int numEmployees;

    // Ask the user how many employees to enter
    printf("Enter the number of employees: ");
    scanf("%d", &numEmployees);

    // Declare an array of Employee structures
    Employee employees[numEmployees];

    // Get employee data from the user
    for (int i = 0; i < numEmployees; i++) {
        printf("\nEnter details for employee %d:\n", i + 1);
        inputEmployeeData(&employees[i]);
    }

    // Display the data of all employees
    printf("\nEmployee Information:\n");
    for (int i = 0; i < numEmployees; i++) {
        displayEmployeeData(employees[i]);
```

```
    }

    return 0;
}
```

2.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>

// Define a union to store different types of data
union Data {
    int i;
    float f;
    char str[20];
};

// Define a structure that contains a nested union
typedef struct {
    int type;    // Type to specify what kind of data is stored
    union Data data; // Nested union
} DataEntry;

// Function to display the contents of the DataEntry structure
void displayData(DataEntry entry) {
    if (entry.type == 1) {
        printf("Integer: %d\n", entry.data.i);
    } else if (entry.type == 2) {
        printf("Float: %.2f\n", entry.data.f);
    } else if (entry.type == 3) {
        printf("String: %s\n", entry.data.str);
    } else {
        printf("Unknown type\n");
    }
}

int main() {
    // Create an array of DataEntry to store different types of data
    DataEntry entries[3];

    // Store different types of data in the array
    entries[0].type = 1;        // Integer type
    entries[0].data.i = 10;

    entries[1].type = 2;        // Float type
    entries[1].data.f = 3.14;

    entries[2].type = 3;        // String type
    strcpy(entries[2].data.str, "Hello, World!");

    // Display the data
    for (int i = 0; i < 3; i++) {
        displayData(entries[i]);
```

```
    }

    return 0;
}
```

3.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 MAX_NAMES 5
#define MAX_LENGTH 100

// Union to store either the length or the reversed string
typedef union {
    int length;
    char reversed[MAX_LENGTH];
} StringInfo;

// Structure to store a name and its associated StringInfo
typedef struct {
    char name[MAX_LENGTH];
    StringInfo info;
    int isLength;  // Flag to indicate whether we store 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
}

int main() {
    NameInfo names[MAX_NAMES];
    int choice;

    // Input names
    printf("Enter %d names:\n", MAX_NAMES);
    for (int i = 0; i < MAX_NAMES; i++) {
        printf("Name %d: ", i + 1);
        scanf("%s", names[i].name);  // Read the name using scanf

        // Ask user to choose between length or reversed string
        printf("Do you want to store (1) length or (2) reversed string for '%s'? ", names[i].name);
        scanf("%d", &choice);

        if (choice == 1) {
            names[i].info.length = strlen(names[i].name);
            names[i].isLength = 1;
        } else if (choice == 2) {
```

```
            reverseString(names[i].name, names[i].info.reversed);
            names[i].isLength = 0;
        } else {
            printf("Invalid choice. Defaulting to length.\n");
            names[i].info.length = strlen(names[i].name);
            names[i].isLength = 1;
        }
    }

    // Display the stored information
    printf("\nStored Information:\n");
    for (int i = 0; i < MAX_NAMES; i++) {
        printf("Name: %s\n", names[i].name);
        if (names[i].isLength) {
            printf("Length: %d\n", names[i].info.length);
        } else {
            printf("Reversed: %s\n", names[i].info.reversed);
        }
        printf("\n");
    }

    return 0;
}
```

4.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>
#include <string.h>

// Define the inner structure
typedef struct {
    int id;
    char name[50];
} Employee;

// Define the union that contains an integer or an Employee structure
typedef union {
    int empId;
    Employee emp;
} EmpUnion;

// Define the outer structure that contains the union
typedef struct {
    char department[50];
    EmpUnion empInfo;  // Union inside the structure
} DepartmentInfo;

int main() {
    // Create an array of DepartmentInfo to manage multiple instances
    DepartmentInfo departments[2];

    // First department with only employee ID
    departments[0].empInfo.empId = 101;
```

```
    strcpy(departments[0].department, "HR");

    // Second department with full employee details
    departments[1].empInfo.emp.id = 202;
    strcpy(departments[1].empInfo.emp.name, "John Doe");
    strcpy(departments[1].department, "Finance");

    // Print out the information
    for (int i = 0; i < 2; i++) {
        printf("Department: %s\n", departments[i].department);

        if (departments[i].empInfo.empId != 0) {
            printf("Employee ID: %d\n", departments[i].empInfo.empId);
        } else {
            printf("Employee ID: N/A\n");
            printf("Employee Name: %s\n", departments[i].empInfo.emp.name);
            printf("Employee ID: %d\n", departments[i].empInfo.emp.id);
        }
        printf("\n");
    }

    return 0;
}
```

5.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>

// Defining a structure for Author's details
typedef struct {
    char name[50];
    char nationality[50];
    int birthYear;
} Author;

// Defining a union to store either the number of pages or the publication year
typedef union {
    int pages;
    int publicationYear;
} BookDetails;

// Defining the main structure for Book
typedef struct {
    char title[100];
    Author author;
    BookDetails details;
    int isPages; // Flag to determine if details refer to pages (1) or publication year (0)
} Book;

// Function to input Author details
void inputAuthor(Author *author) {
    printf("Enter author's name: ");
```

```
    scanf("%49s", author->name); // Limiting the input size to avoid buffer overflow

    printf("Enter author's nationality: ");
    scanf("%49s", author->nationality); // Limiting the input size to avoid buffer overflow

    printf("Enter author's birth year: ");
    scanf("%d", &author->birthYear);
}

// Function to input Book details
void inputBook(Book *book) {
    printf("Enter book title: ");
    scanf("%99s", book->title); // Limiting the input size to avoid buffer overflow

    // Input author's details
    inputAuthor(&book->author);

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

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

// Function to display Author details
void displayAuthor(const Author *author) {
    printf("Author Name: %s\n", author->name);
    printf("Author Nationality: %s\n", author->nationality);
    printf("Author Birth Year: %d\n", author->birthYear);
}

// Function to display Book details
void displayBook(const Book *book) {
    printf("\nBook Title: %s\n", book->title);
    displayAuthor(&book->author);

    if (book->isPages == 1) {
        printf("Number of Pages: %d\n", book->details.pages);
    } else {
        printf("Publication Year: %d\n", book->details.publicationYear);
    }
}

int main() {
    Book myBook;

    // Input Book details
    inputBook(&myBook);
```

```
    // Display Book details
    displayBook(&myBook);

    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

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

#define MAX 5  // Define maximum size of the stack

// Stack structure
struct Stack {
    int arr[MAX];
    int top;
};

// Function to initialize the stack
void initStack(struct Stack *s) {
    s->top = -1;  // Indicates that the stack is empty
}

// Function to check if the stack is empty
int isEmpty(struct Stack *s) {
    return s->top == -1;
}

// Function to check if the stack is full
int isFull(struct Stack *s) {
    return s->top == MAX - 1;
}

// Function to push an element onto the stack
void push(struct Stack *s, int value) {
    if (isFull(s)) {
        printf("Stack Overflow! Unable to push %d\n", value);
    } else {
        s->arr[++(s->top)] = value;
        printf("Pushed %d to the stack\n", value);
    }
}

// Function to pop an element from the stack
int pop(struct Stack *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow! Unable to pop\n");
        return -1;  // Return -1 to indicate stack underflow
    } else {
        int poppedValue = s->arr[(s->top)--];
```

```
        printf("Popped %d from the stack\n", poppedValue);
        return poppedValue;
    }
}

// Function to peek the top element of the stack
int peek(struct Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty! No elements to peek\n");
        return -1;  // Return -1 if the stack is empty
    } else {
        return s->arr[s->top];
    }
}

// Function to display the contents of the stack
void display(struct Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty!\n");
    } else {
        printf("Stack elements: ");
        for (int i = 0; i <= s->top; i++) {
            printf("%d ", s->arr[i]);
        }
        printf("\n");
    }
}

// Main function to demonstrate the stack operations
int main() {
    struct Stack s;
    initStack(&s);  // Initialize the stack

    // Demonstrate push operation
    push(&s, 10);
    push(&s, 20);
    push(&s, 30);
    push(&s, 40);
    push(&s, 50);
    push(&s, 60);  // This will show stack overflow since the max size is 5

    // Display the stack
    display(&s);

    // Demonstrate peek operation
    printf("Top element is %d\n", peek(&s));

    // Demonstrate pop operation
    pop(&s);
    pop(&s);

    // Display the stack after popping elements
    display(&s);

    // Peek after popping
```

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

    // Try popping all elements to cause underflow
    pop(&s);
    pop(&s);
    pop(&s);
    pop(&s);  // This will show stack underflow

    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.

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

// Define the structure for the node of the stack
struct Node {
    int data;
    struct Node* next;
};

// Define the structure for the stack
struct Stack {
    struct Node* top;
};

// Function to initialize the stack
void initStack(struct Stack* stack) {
    stack->top = NULL;
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

// Function to push an element onto the stack
void push(struct Stack* stack, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

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

    newNode->data = value;
    newNode->next = stack->top;
    stack->top = newNode;
}

// Function to pop an element from the stack
int pop(struct Stack* stack) {
```

```
    if (isEmpty(stack)) {
        printf("Stack underflow! The stack is empty.\n");
        return -1; // Return an error value
    }

    struct Node* temp = stack->top;
    int poppedValue = temp->data;
    stack->top = stack->top->next;
    free(temp);

    return poppedValue;
}

// Function to peek the top element of the stack
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
        return -1; // Return an error value
    }

    return stack->top->data;
}

// Function to check if the stack is full
// For a linked list implementation, we cannot know the maximum size directly
// so we assume a very large value for the purpose of this demonstration.
int isFull(struct Stack* stack) {
    // A stack using a linked list is never technically "full" unless memory runs out.
    // We could return 0 here as a placeholder for stack being never full in terms of size.
    return 0; // Return 0 because the stack can dynamically grow.
}

// Function to free the entire stack's memory
void freeStack(struct Stack* stack) {
    while (!isEmpty(stack)) {
        pop(stack);
    }
}

// Main function to test the stack operations
int main() {
    struct Stack stack;
    initStack(&stack);

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

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

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

    printf("Is stack empty? %d\n", isEmpty(&stack));
```

```
    push(&stack, 40);
    printf("Top element after push: %d\n", peek(&stack));

    // Clean up the stack memory
    freeStack(&stack);

    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.

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

#define MAX 10  // Maximum size of the stack

// Define the stack structure
typedef struct {
    int arr[MAX];
    int top;
} Stack;

// Function to initialize the stack
void initStack(Stack* stack) {
    stack->top = -1;  // Stack is initially empty
}

// Function to check if the stack is full
int isFull(Stack* stack) {
    return stack->top == MAX - 1;
}

// Function to check if the stack is empty
int isEmpty(Stack* stack) {
    return stack->top == -1;
}

// Function to push an element onto the stack
void push(Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack Overflow! Cannot push %d\n", value);
    } else {
        stack->arr[++(stack->top)] = value;
        printf("Pushed %d onto the stack.\n", value);
    }
}

// Function to pop an element from the stack
int pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow! Cannot pop from the stack.\n");
        return -1;
    } else {
        return stack->arr[(stack->top)--];
```

```
    }
}

// Function to peek the top element of the stack
int peek(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
        return -1;
    } else {
        return stack->arr[stack->top];
    }
}

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

// Function to reverse the stack
void reverseStack(Stack* stack) {
    int tempStack[MAX];
    int tempTop = -1;

    // Pop elements from the original stack and push them into tempStack
    while (!isEmpty(stack)) {
        tempStack[++tempTop] = pop(stack);
    }

    // Push the elements from tempStack back to the original stack
    for (int i = tempTop; i >= 0; i--) {
        push(stack, tempStack[i]);
    }

    printf("Stack reversed successfully.\n");
}

// Main function to demonstrate the stack operations
int main() {
    Stack stack;
    initStack(&stack);

    // Demonstrate push operation
    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    push(&stack, 40);
    push(&stack, 50);
```

```
    // Display current stack
    display(&stack);

    // Reverse the stack
    reverseStack(&stack);

    // Display the reversed stack
    display(&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.

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

// Stack node definition
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new stack node
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->next = NULL;
    return node;
}

// Function to push an element onto the stack
void push(struct Node** top, int data) {
    struct Node* node = newNode(data);
    node->next = *top;
    *top = node;
}

// Function to pop an element from the stack
int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack is empty\n");
        return -1;  // Return a sentinel value
    }
    struct Node* temp = *top;
    int poppedValue = temp->data;
    *top = (*top)->next;
    free(temp);
    return poppedValue;
}

// Function to display the stack
```

```
void display(struct Node* top) {
    if (top == NULL) {
        printf("Stack is empty\n");
        return;
    }
    struct Node* temp = top;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to merge two stacks
void mergeStacks(struct Node** stack1, struct Node** stack2) {
    // If the first stack is empty, point it to the second stack
    if (*stack1 == NULL) {
        *stack1 = *stack2;
        *stack2 = NULL;
        return;
    }

    // Otherwise, merge stack2 into stack1 by traversing to the last element of stack1
    struct Node* temp = *stack1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = *stack2;
    *stack2 = NULL;
}

int main() {
    struct Node* stack1 = NULL;
    struct Node* stack2 = NULL;

    // Push elements onto stack1
    push(&stack1, 10);
    push(&stack1, 20);
    push(&stack1, 30);

    // Push elements onto stack2
    push(&stack2, 40);
    push(&stack2, 50);

    printf("Stack 1 before merge:\n");
    display(stack1);
    printf("Stack 2 before merge:\n");
    display(stack2);

    // Merge stack2 into stack1
    mergeStacks(&stack1, &stack2);

    printf("\nStack 1 after merge:\n");
    display(stack1);
    printf("Stack 2 after merge (should be empty):\n");
```

```
    display(stack2);

    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.

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

// Define the maximum size of the stack
#define MAX 100

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

// Function to initialize the stack
void initStack(struct Stack *stack) {
    stack->top = -1;
}

// Function to check if the stack is empty
bool isEmpty(struct Stack *stack) {
    return stack->top == -1;
}

// Function to check if the stack is full
bool isFull(struct Stack *stack) {
    return stack->top == MAX - 1;
}

// Function to push an element to the stack
void push(struct Stack *stack, char c) {
    if (isFull(stack)) {
        printf("Stack Overflow\n");
        return;
    }
    stack->arr[++(stack->top)] = c;
}

// Function to pop an element from the stack
char pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack->arr[(stack->top)--];
}

// Function to get the top element of the stack
```

```
char peek(struct Stack *stack) {
    if (isEmpty(stack)) {
        return -1;
    }
    return stack->arr[stack->top];
}

// Function to check if the parentheses are balanced
bool isBalanced(char *exp) {
    struct Stack stack;
    initStack(&stack);

    // Traverse through the expression
    for (int i = 0; exp[i] != '\0'; i++) {
        char currentChar = exp[i];

        // If it's an opening parenthesis, push it to the stack
        if (currentChar == '(') {
            push(&stack, currentChar);
        }
        // If it's a closing parenthesis, check if it matches the top of the stack
        else if (currentChar == ')') {
            if (isEmpty(&stack)) {
                return false; // Unmatched closing parenthesis
            }
            pop(&stack); // Pop the matched opening parenthesis
        }
    }

    // If the stack is empty, parentheses are balanced
    return isEmpty(&stack);
}

// Main function to demonstrate the balanced parentheses check
int main() {
    char expression[100];

    // Get the expression from the user
    printf("Enter an expression with parentheses: ");
    fgets(expression, sizeof(expression), stdin);

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

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

// Define a node for the linked list
struct Node {
    int data;
    struct Node* next;
};

// Define a Stack
struct Stack {
    struct Node* top;
};

// Function to create an empty stack
struct Stack* createStack() {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = NULL;
    return stack;
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

// Function to push an element onto the stack
void push(struct Stack* stack, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = stack->top;
    stack->top = newNode;
}

// Function to pop an element from the stack
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
        exit(1);
    }
    struct Node* temp = stack->top;
    int value = temp->data;
    stack->top = stack->top->next;
    free(temp);
    return value;
}

// Function to peek the top element of the stack
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
        exit(1);
    }
    return stack->top->data;
```

```
}

// Function to perform arithmetic operations
int performOperation(int operand1, int operand2, char operator) {
    switch (operator) {
        case '+': return operand1 + operand2;
        case '-': return operand1 - operand2;
        case '*': return operand1 * operand2;
        case '/': return operand1 / operand2;
        default:
            printf("Invalid operator!\n");
            exit(1);
    }
}

// Function to evaluate a postfix expression
int evaluatePostfix(char* expression) {
    struct Stack* stack = createStack();
    int i = 0;

    while (expression[i] != '\0') {
        // Skip spaces in the expression
        if (expression[i] == ' ') {
            i++;
            continue;
        }

        // If the character is a digit, push it onto the stack
        if (isdigit(expression[i])) {
            int num = 0;
            // Convert the digit from char to int
            while (isdigit(expression[i])) {
                num = num * 10 + (expression[i] - '0');
                i++;
            }
            push(stack, num);
        }
        // If the character is an operator, pop two operands and apply the operator
        else if (expression[i] == '+' || expression[i] == '-' || expression[i] == '*' || expression[i] == '/') {
            int operand2 = pop(stack);
            int operand1 = pop(stack);
            int result = performOperation(operand1, operand2, expression[i]);
            push(stack, result);
            i++;
        }
        else {
            printf("Invalid character in expression: %c\n", expression[i]);
            exit(1);
        }
    }

    // The result should be the only element left in the stack
    int result = pop(stack);
    if (!isEmpty(stack)) {
        printf("Invalid expression!\n");
```

```
        exit(1);
    }

    return result;
}

int main() {
    char expression[100];

    // Input postfix expression
    printf("Enter a postfix expression: ");
    fgets(expression, sizeof(expression), stdin);

    // Remove newline character if present
    if (expression[strlen(expression) - 1] == '\n') {
        expression[strlen(expression) - 1] = '\0';
    }

    // Evaluate the expression
    int result = evaluatePostfix(expression);

    // Output the result
    printf("Result: %d\n", result);

    return 0;
}
```