```c
// Write a program in c to search the element using binary search.

#include <stdio.h>

// Function to perform binary search
int binarySearch(int arr[], int size, int key) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == key) {
            return mid; // Return the index where the key is found
        }

        if (arr[mid] < key) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

    return -1; // Return -1 if the key is not found
}

int main() {
    int arr[] = {11, 12, 22, 25, 34, 64, 90};
    int size = sizeof(arr) / sizeof(arr[0]);

    int key;
    printf("Enter the element to search for: ");
    scanf("%d", &key);

    int result = binarySearch(arr, size, key);

    if (result != -1) {
        printf("%d found at index %d\n", key, result);
    } else {
        printf("%d not found in the array\n", key);
    }

    return 0;
}
/*
Output
Enter the element to search for: 25
25 found at index 3
*/
```

```c
// Write a program in c to implement bubble sort.
#include <stdio.h>
void bubbleSort(int arr[], int size) {
    int temp;
    int swapped;
    for (int i = 0; i < size - 1; i++) {
        swapped = 0; // Flag to check if any swaps were made in this pass

        for (int j = 0; j < size - 1 - i; j++) {
            // If the current element is greater than the next element, swap them
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = 1; // Set the swapped flag to 1
            }
        }

        // If no two elements were swapped in this pass, the array is already sorted
        if (swapped == 0) {
            break;
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    // Perform bubble sort
    bubbleSort(arr, size);

    printf("\nSorted array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
/*
Output
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
*/
```

```c
// Que:- Write a program in c to create double linked list and display the elements in reverse order.
#include <stdio.h>
#include <stdlib.h>

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

// Function to insert a new node at the end of the doubly linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}

// Function to display the doubly linked list in reverse order
void displayReverse(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
```

```c
        return;
    }

    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }

    printf("Doubly Linked List (in reverse order): ");
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->prev;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int n, data;

    printf("Enter the number of elements in the doubly linked list: ");
    scanf("%d", &n);

    printf("Enter the elements of the doubly linked list:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        insertAtEnd(&head, data);
    }

    displayReverse(head);

    return 0;
}

/*
Output
Enter the number of elements in the doubly linked list: 3
Enter the elements of the doubly linked list:
3
2
1
Doubly Linked List (in reverse order): 1 -> 2 -> 3 -> NULL

*/
```

```c
// Write a program in c to search the element using sequential/linear search.

#include <stdio.h>

// Function to perform sequential (linear) search
int linearSearch(int arr[], int size, int key) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            return i; // Return the index where the key is found
        }
    }
    return -1; // Return -1 if the key is not found
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);

    int key;
    printf("Enter the element to search for: ");
    scanf("%d", &key);

    int result = linearSearch(arr, size, key);

    if (result != -1) {
        printf("%d found at index %d\n", key, result);
    } else {
        printf("%d not found in the array\n", key);
    }

    return 0;
}


/*
Output
Enter the element to search for: 34
34 found at index 1
*/
```

```c
// Que:-  Write a program in c to implement the concept of Circular Queue
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 5

// Structure to represent the Circular Queue
struct CircularQueue {
    int items[MAX_SIZE];
    int front, rear;
};

// Function to enqueue an element into the Circular Queue
void enqueue(struct CircularQueue* queue, int data) {
    if (((queue->rear + 1) % MAX_SIZE) == queue->front) {
        printf("Queue is full. Cannot enqueue %d\n", data);
    } else {
        if (queue->front == -1) {
            queue->front = 0;
        }
        queue->rear = (queue->rear + 1) % MAX_SIZE;
        queue->items[queue->rear] = data;
        printf("%d enqueued to the queue\n", data);
    }
}

// Function to dequeue an element from the Circular Queue
int dequeue(struct CircularQueue* queue) {
    int data = -1;
    if (queue->front == -1) {
        printf("Queue is empty. Cannot dequeue\n");
    } else {
        data = queue->items[queue->front];
        if (queue->front == queue->rear) {
```

```c
            // Queue has only one element, reset front and rear
            queue->front = -1;
            queue->rear = -1;
        } else {
            queue->front = (queue->front + 1) % MAX_SIZE;
        }
    }
    return data;
}

// Function to display the elements in the Circular Queue
void display(struct CircularQueue* queue) {
    if (queue->front == -1) {
        printf("Queue is empty\n");
    } else {
        int i = queue->front;
        printf("Queue elements: ");
        while (1) {
            printf("%d ", queue->items[i]);
            if (i == queue->rear) {
                break;
            }
            i = (i + 1) % MAX_SIZE;
        }
        printf("\n");
    }
}

int main() {
    struct CircularQueue queue;
    queue.front = -1;
    queue.rear = -1;
```

```c
    int choice, data;

    do {
        printf("\nCircular Queue Menu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(&queue, data);
                break;
            case 2:
                data = dequeue(&queue);
                if (data != -1) {
                    printf("Dequeued element: %d\n", data);
                }
                break;
            case 3:
                display(&queue);
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 4);
```

```
    return 0;
}
/*
Output

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 1
Enter data to enqueue: 1
1 enqueued to the queue

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 3
Queue elements: 1

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 1
Enter data to enqueue: 2
2 enqueued to the queue

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 3
Queue elements: 1 2

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 2
Dequeued element: 1

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 3
Queue elements: 2

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice:
```

```c
// Write a program in c to implement quick sort.

#include <stdio.h>

// Function to swap two elements in an array
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function to partition the array and return the pivot index
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Choose the last element as the pivot
    int i = (low - 1); // Index of the smaller element

    for (int j = low; j <= high - 1; j++) {
        // If the current element is smaller than or equal to the pivot
        if (arr[j] <= pivot) {
            i++; // Increment the index of the smaller element
            swap(&arr[i], &arr[j]);
        }
    }

    // Swap the pivot element with the element at (i+1), placing the pivot in its correct position
    swap(&arr[i + 1], &arr[high]);
    return (i + 1); // Return the pivot index
}

// Function to perform quick sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Find the pivot element such that
        // elements smaller than the pivot are on the left
        // elements greater than the pivot are on the right
        int pi = partition(arr, low, high);

        // Recursively sort elements before and after the pivot
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    // Perform quick sort
    quickSort(arr, 0, size - 1);

    printf("\nSorted array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
/*
Output
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
*/
```

```c
// Que:- Write a program in c to search the elements in the linked list and display the position

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

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

// Function to search for an element in the linked list and return its position
int searchElement(struct Node* head, int key) {
    int position = 1;
    while (head != NULL) {
        if (head->data == key) {
            return position;
        }
        head = head->next;
        position++;
    }
    return -1; // Element not found
}

// Function to insert a new node at the beginning of the linked list
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

int main() {
    struct Node* head = NULL;
    int n, key;

    printf("Enter the number of elements in the linked list: ");
    scanf("%d", &n);

    printf("Enter the elements of the linked list:\n");
    for (int i = 0; i < n; i++) {
        int data;
        scanf("%d", &data);
        insertAtBeginning(&head, data);
    }

    printf("Enter the element to search: ");
    scanf("%d", &key);

    int position = searchElement(head, key);
```

```c
    printf("Enter the elements of the linked list:\n");
    for (int i = 0; i < n; i++) {
        int data;
        scanf("%d", &data);
        insertAtBeginning(&head, data);
    }

    printf("Enter the element to search: ");
    scanf("%d", &key);

    int position = searchElement(head, key);
    if (position != -1) {
        printf("Element found at position %d\n", position);
    } else {
        printf("Element not found in the linked list\n");
    }

    return 0;
}

/*
OutPut
Enter the number of elements in the linked list: 8
Enter the elements of the linked list:
1
2
3
5
6
4
9
7
Enter the element to search: 5
Element found at position 5
*/
```

```c
// Write a program in c to implement the Stack with Push, Pop and Display operations.
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

struct Stack {
    int items[MAX_SIZE];
    int top;
};

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

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

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

void push(struct Stack *stack, int value) {
    if (isFull(stack)) {
        printf("Stack is full. Cannot push %d\n", value);
    } else {
        stack->top++;
        stack->items[stack->top] = value;
        printf("%d pushed onto the stack\n", value);
    }
}

int pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. Cannot pop.\n");
        return -1;
    } else {
        int poppedValue = stack->items[stack->top];
        stack->top--;
        return poppedValue;
```

```c
    }
}

void display(struct 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->items[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Stack stack;
    initialize(&stack);

    int choice, value;

    while (1) {
        printf("\nStack Menu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 2:
                printf("Popped element: %d\n", pop(&stack));
                break;
            case 3:
                display(&stack);
```

```c
                display(&stack);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
/*
OutPut

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack is empty.

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 1
1 pushed onto the stack

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 2
2 pushed onto the stack

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 3
3 pushed onto the stack

Stack Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 1 2 3
```

traversal of tree.c

```c
// Write a program in c for in order, post order and preorder traversal of tree

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

// Define the structure for a binary tree node
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

// Function to create a new binary tree node
struct TreeNode* createNode(int data) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function for in-order traversal of a binary tree
void inorderTraversal(struct TreeNode* root) {
    if (root == NULL) {
        return;
    }
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

// Function for post-order traversal of a binary tree
void postorderTraversal(struct TreeNode* root) {
    if (root == NULL) {
        return;
    }
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ", root->data);
}

// Function for pre-order traversal of a binary tree
void preorderTraversal(struct TreeNode* root) {
    if (root == NULL) {
        return;
    }
    printf("%d ", root->data);
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

int main() {
    // Create a sample binary tree
    struct TreeNode* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);

    printf("In-order traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Post-order traversal: ");
    postorderTraversal(root);
    printf("\n");

    printf("Pre-order traversal: ");
    preorderTraversal(root);
    printf("\n");

    return 0;
}
/*
Output

In-order traversal: 4 2 5 1 3
Post-order traversal: 4 5 2 3 1
Pre-order traversal: 1 2 4 5 3

*/
```

```c
//Que:- Write a program in c to read two arrays from the user and merge them and display the elements
#include <stdio.h>

int main() {
    int arr1[100], arr2[100], mergeArr[200];
    int size1, size2, mergedSize;

    // Input for the first array
    printf("Enter the size of the first array: ");
    scanf("%d", &size1);

    printf("Enter elements of the first array:\n");
    for (int i = 0; i < size1; i++) {
        scanf("%d", &arr1[i]);
    }

    // Input for the second array
    printf("Enter the size of the second array: ");
    scanf("%d", &size2);

    printf("Enter elements of the second array:\n");
    for (int i = 0; i < size2; i++) {
        scanf("%d", &arr2[i]);
    }

    // Merging the two arrays
    mergedSize = size1 + size2;
    for (int i = 0; i < size1; i++) {
        mergeArr[i] = arr1[i];
    }
    for (int i = 0; i < size2; i++) {
        mergeArr[size1 + i] = arr2[i];
    }

    // Displaying the merged array
    printf("Merged array: ");
    for (int i = 0; i < mergedSize; i++) {
        printf("%d ", mergeArr[i]);
    }
    printf("\n");
```

```c
    printf("Merged array: ");
    for (int i = 0; i < mergedSize; i++) {
        printf("%d ", mergeArr[i]);
    }
    printf("\n");

    return 0;
}

// output
/*
Enter the size of the first array: 3
Enter elements of the first array:
1
2
3
Enter the size of the second array: 3
Enter elements of the second array:
9
8
7
Merged array: 1 2 3 9 8 7

*/
```