Q1. WAP TO IMPLEMENT STACK USING ARRAY OF SIZE 5. USING PUSH OPERATION FIVE TIMES PUSH 5, 4, 3, 2, 1. DISPLAY THE RESULT. POP ONE ELEMENT. DISPLAY THE RESULT.

Ans1: Sample C program:

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

void push(int stack[], int *top, int value) {
    if (*top == SIZE - 1) {
        printf("Stack overflow\n");
        return;
    }
    (*top)++;
    stack[*top] = value;
}

int pop(int stack[], int *top) {
    if (*top == -1) {
        printf("Stack underflow\n");
        return -1;
    }
    return stack[(*top)--];
}

void display(int stack[], int top) {
    int i;
    if (top == -1) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack (top to bottom): ");
    for (i = top; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}

int main() {
    int stack[SIZE];
    int top = -1;
    int popped;

    // push 5, 4, 3, 2, 1
    push(stack, &top, 5);
    push(stack, &top, 4);
    push(stack, &top, 3);
    push(stack, &top, 2);
    push(stack, &top, 1);

    printf("After pushing 5,4,3,2,1:\n");
    display(stack, top);
```

```
    // pop one element
    popped = pop(stack, &top);
    printf("Popped element: %d\n", popped);
    printf("After one pop:\n");
    display(stack, top);

    return 0;
}
```

Q2. WAP TO IMPLEMENT STACK USING ARRAY OF SIZE 5. USING PUSH OPERATION FIVE TIMES PUSH 10, 9, 8, 7, 6. DISPLAY THE RESULT. POP ONE ELEMENT. DISPLAY THE RESULT.

Ans2: Sample C program:

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

void push(int stack[], int *top, int value) {
    if (*top == SIZE - 1) {
        printf("Stack overflow\n");
        return;
    }
    (*top)++;
    stack[*top] = value;
}

int pop(int stack[], int *top) {
    if (*top == -1) {
        printf("Stack underflow\n");
        return -1;
    }
    return stack[(*top)--];
}

void display(int stack[], int top) {
    int i;
    if (top == -1) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stack (top to bottom): ");
    for (i = top; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}

int main() {
    int stack[SIZE];
    int top = -1;
    int popped;
```

```
    // push 10, 9, 8, 7, 6
    push(stack, &top, 10);
    push(stack, &top, 9);
    push(stack, &top, 8);
    push(stack, &top, 7);
    push(stack, &top, 6);

    printf("After pushing 10,9,8,7,6:\n");
    display(stack, top);

    // pop one element
    popped = pop(stack, &top);
    printf("Popped element: %d\n", popped);
    printf("After one pop:\n");
    display(stack, top);

    return 0;
}
```

Q3. WAP TO IMPLEMENT STACK USING LINKED LIST. USING PUSH OPERATION FIVE TIMES PUSH 5, 4, 3, 2, 1. DISPLAY THE RESULT. POP ONE ELEMENT. DISPLAY THE RESULT.

Ans3: Sample C program:

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

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

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

int pop(struct Node **top) {
    if (*top == NULL) {
        printf("Stack underflow\n");
        return -1;
    }
    struct Node *temp = *top;
    int value = temp->data;
    *top = temp->next;
    free(temp);
    return value;
}
void display(struct Node *top) {
```

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

int main() {
    struct Node *top = NULL;
    int popped;

    // push 5,4,3,2,1
    push(&top, 5);
    push(&top, 4);
    push(&top, 3);
    push(&top, 2);
    push(&top, 1);

    printf("After pushing 5,4,3,2,1:\n");
    display(top);

    // pop one element
    popped = pop(&top);
    printf("Popped element: %d\n", popped);
    printf("After one pop:\n");
    display(top);

    return 0;
}
```

Q4. WAP TO IMPLEMENT STACK USING LINKED LIST. USING PUSH OPERATION FIVE TIMES PUSH 10, 9, 8, 7, 6. DISPLAY THE RESULT. POP ONE ELEMENT. DISPLAY THE RESULT.

Ans4: Sample C program:

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

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

void push(struct Node **top, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
```

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

int pop(struct Node **top) {
    if (*top == NULL) {
        printf("Stack underflow\n");
        return -1;
    }
    struct Node *temp = *top;
    int value = temp->data;
    *top = temp->next;
    free(temp);
    return value;
}

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

int main() {
    struct Node *top = NULL;
    int popped;

    // push 10,9,8,7,6
    push(&top, 10);
    push(&top, 9);
    push(&top, 8);
    push(&top, 7);
    push(&top, 6);

    printf("After pushing 10,9,8,7,6:\n");
    display(top);

    // pop one element
    popped = pop(&top);
    printf("Popped element: %d\n", popped);
    printf("After one pop:\n");
    display(top);

    return 0;
}
```

Q5. WAP TO IMPLEMENT QUEUE USING ARRAY OF SIZE 5. USING ENQUEUE OPERATION FIVE TIMES INSERT 10, 9, 8, 7, 6. DISPLAY THE RESULT. DEQUEUE ONE ELEMENT. DISPLAY THE RESULT.

Ans5: Sample C program:

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

void enqueue(int queue[], int *rear, int *front, int value) {
    if (*rear == SIZE - 1) {
        printf("Queue overflow\n");
        return;
    }
    if (*front == -1) {
        *front = 0;
    }
    (*rear)++;
    queue[*rear] = value;
}

int dequeue(int queue[], int *rear, int *front) {
    int value;
    if (*front == -1 || *front > *rear) {
        printf("Queue underflow\n");
        return -1;
    }
    value = queue[*front];
    (*front)++;
    return value;
}

void display(int queue[], int rear, int front) {
    int i;
    if (front == -1 || front > rear) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue (front to rear): ");
    for (i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    int queue[SIZE];
    int front = -1, rear = -1;
    int removed;

    // enqueue 10,9,8,7,6
```

```
    enqueue(queue, &rear, &front, 10);
    enqueue(queue, &rear, &front, 9);
    enqueue(queue, &rear, &front, 8);
    enqueue(queue, &rear, &front, 7);
    enqueue(queue, &rear, &front, 6);

    printf("After enqueuing 10,9,8,7,6:\n");
    display(queue, rear, front);

    // dequeue one element
    removed = dequeue(queue, &rear, &front);
    printf("Dequeued element: %d\n", removed);
    printf("After one dequeue:\n");
    display(queue, rear, front);

    return 0;
}
```

Q6. WAP TO IMPLEMENT QUEUE USING ARRAY OF SIZE 5. USING ENQUEUE OPERATION FIVE TIMES INSERT 5, 4, 3, 2, 1. DISPLAY THE RESULT. DEQUEUE ONE ELEMENT. DISPLAY THE RESULT.

Ans6: Sample C program:

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

void enqueue(int queue[], int *rear, int *front, int value) {
    if (*rear == SIZE - 1) {
        printf("Queue overflow\n");
        return;
    }
    if (*front == -1) {
        *front = 0;
    }
    (*rear)++;
    queue[*rear] = value;
}

int dequeue(int queue[], int *rear, int *front) {
    int value;
    if (*front == -1 || *front > *rear) {
        printf("Queue underflow\n");
        return -1;
    }
    value = queue[*front];
    (*front)++;
    return value;
}

void display(int queue[], int rear, int front) {
    int i;
```

```
    if (front == -1 || front > rear) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue (front to rear): ");
    for (i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    int queue[SIZE];
    int front = -1, rear = -1;
    int removed;

    // enqueue 5,4,3,2,1
    enqueue(queue, &rear, &front, 5);
    enqueue(queue, &rear, &front, 4);
    enqueue(queue, &rear, &front, 3);
    enqueue(queue, &rear, &front, 2);
    enqueue(queue, &rear, &front, 1);

    printf("After enqueuing 5,4,3,2,1:\n");
    display(queue, rear, front);

    // dequeue one element
    removed = dequeue(queue, &rear, &front);
    printf("Dequeued element: %d\n", removed);
    printf("After one dequeue:\n");
    display(queue, rear, front);

    return 0;
}
```

Q7. WAP TO IMPLEMENT LINKED LIST. USING INSERTING NEW NODE AT END AND INSERTING NEW NODE AT FIRST CREATE LINKED LIST 1->2->3->4->5. USING TRAVERSING DISPLAY THE LIST.

Ans7: Sample C program:

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

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

void insertAtEnd(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
```

```
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node *temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void insertAtFirst(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void display(struct Node *head) {
    struct Node *temp = head;
    printf("Linked list: ");
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) printf("->");
        temp = temp->next;
    }
    printf("\n");
}


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

    // Example: create 1->2->3->4->5
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 4);
    insertAtEnd(&head, 5);
    insertAtFirst(&head, 1);

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

Q8. WAP TO IMPLEMENT LINKED LIST. USING INSERTING NEW NODE AT END AND INSERTING NEW NODE AT FIRST CREATE LINKED LIST 10->20->30->40. USING TRAVERSING DISPLAY THE LIST.

Ans8: Sample C program:

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

```c
#include <stdlib.h>

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

void insertAtEnd(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node *temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void insertAtFirst(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void display(struct Node *head) {
    struct Node *temp = head;
    printf("Linked list: ");
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) printf("->");
        temp = temp->next;
    }
    printf("\n");
}


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

    // Example: create 10->20->30->40
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    insertAtEnd(&head, 40);
    insertAtFirst(&head, 10);

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

Q9. WAP TO IMPLEMENT LINKED LIST. USING INSERTING NEW NODE AT END CREATE LINKED LIST 1->2->3->4->5. USING TRAVERSING DISPLAY THE LIST. USE DELETION AT END ONCE, DISPLAY THE UPDATED LIST.

Ans9: Sample C program:

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

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

void insertAtEnd(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node *temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void insertAtFirst(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void display(struct Node *head) {
    struct Node *temp = head;
    printf("Linked list: ");
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) printf("->");
        temp = temp->next;
    }
    printf("\n");
}


void deleteAtEnd(struct Node **head) {
    if (*head == NULL) return;
    if ((*head)->next == NULL) {
```

```
        free(*head);
        *head = NULL;
        return;
    }
    struct Node *temp = *head;
    struct Node *prev = NULL;
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    prev->next = NULL;
    free(temp);
}

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

    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 4);
    insertAtEnd(&head, 5);

    printf("Original list:\n");
    display(head);

    deleteAtEnd(&head);
    printf("After deleting at end:\n");
    display(head);

    return 0;
}
```

Q10. WAP TO IMPLEMENT LINKED LIST. USING INSERTING NEW NODE AT END CREATE LINKED LIST 1->2->3->4->5. USING TRAVERSING DISPLAY THE LIST. USE DELETION AT FRONT ONCE, DISPLAY THE UPDATED LIST.

Ans10: Sample C program:

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

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

void insertAtEnd(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (*head == NULL) {
```

```
        *head = newNode;
        return;
    }
    struct Node *temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void insertAtFirst(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void display(struct Node *head) {
    struct Node *temp = head;
    printf("Linked list: ");
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) printf("->");
        temp = temp->next;
    }
    printf("\n");
}


void deleteAtFront(struct Node **head) {
    if (*head == NULL) return;
    struct Node *temp = *head;
    *head = (*head)->next;
    free(temp);
}

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

    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 4);
    insertAtEnd(&head, 5);

    printf("Original list:\n");
    display(head);

    deleteAtFront(&head);
    printf("After deleting at front:\n");
    display(head);

    return 0;
```

}

*Q11. WAP TO IMPLEMENT INFIX TO POSTFIX. CONVERT THE FOLLOWING INFIX EXPRESSION TO POSTFIX USING THE PROGRAM. EXPRESSION: (A+B)*C+(D-E).*

Ans11: Sample C program:

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

#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char c) {
    stack[++top] = c;
}

char pop() {
    if (top == -1) return -1;
    return stack[top--];
}

int precedence(char c) {
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
}

int main() {
    char infix[] = "(A+B)*C+(D-E)";
    char postfix[SIZE];
    int i, j = 0;
    char ch, x;

    for (i = 0; infix[i] != '\0'; i++) {
        ch = infix[i];
        if (isalnum(ch)) {
            postfix[j++] = ch;
        } else if (ch == '(') {
            push(ch);
        } else if (ch == ')') {
            while ((x = pop()) != '(') {
                postfix[j++] = x;
            }
        } else {
            while (top != -1 && precedence(stack[top]) >= precedence(ch)) {
                postfix[j++] = pop();
            }
            push(ch);
```

```
        }
    }

    while (top != -1) {
        postfix[j++] = pop();
    }
    postfix[j] = '\0';

    printf("Infix: %s\n", infix);
    printf("Postfix: %s\n", postfix);  // Output: AB+C*DE-+
    return 0;
}
```

Q12. WAP TO IMPLEMENT INFIX TO POSTFIX. CONVERT THE FOLLOWING INFIX EXPRESSION TO POSTFIX USING THE PROGRAM. EXPRESSION: (A-B)-C(D/E).*

Ans12: Sample C program:

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

#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char c) {
    stack[++top] = c;
}

char pop() {
    if (top == -1) return -1;
    return stack[top--];
}

int precedence(char c) {
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
}

int main() {
    char infix[] = "(A-B)-C*(D/E)";
    char postfix[SIZE];
    int i, j = 0;
    char ch, x;

    for (i = 0; infix[i] != '\0'; i++) {
        ch = infix[i];
        if (isalnum(ch)) {
            postfix[j++] = ch;
```

```
        } else if (ch == '(') {
            push(ch);
        } else if (ch == ')') {
            while ((x = pop()) != '(') {
                postfix[j++] = x;
            }
        } else {
            while (top != -1 && precedence(stack[top]) >= precedence(ch)) {
                postfix[j++] = pop();
            }
            push(ch);
        }
    }

    while (top != -1) {
        postfix[j++] = pop();
    }
    postfix[j] = '\0';

    printf("Infix: %s\n", infix);
    printf("Postfix: %s\n", postfix);  // Output: AB+C*DE-+
    return 0;
}
```

Q13. WAP TO IMPLEMENT CIRCULAR DOUBLY LINKED LIST. USING INSERTING NEW NODE AT END CREATE LINKED LIST 1->2->3->4->5. USING TRAVERSING DISPLAY THE LIST. USE DELETION AT FRONT ONCE, DISPLAY THE UPDATED LIST.

Ans13: Sample C program:

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

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

void insertAtEnd(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    if (*head == NULL) {
        newNode->next = newNode;
        newNode->prev = newNode;
        *head = newNode;
        return;
    }
    struct Node *last = (*head)->prev;

    newNode->next = *head;
```

```
    (*head)->prev = newNode;
    newNode->prev = last;
    last->next = newNode;
}

void display(struct Node *head) {
    struct Node *temp;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Circular Doubly Linked List: ");
    temp = head;
    do {
        printf("%d", temp->data);
        temp = temp->next;
        if (temp != head) printf("->");
    } while (temp != head);
    printf("\n");
}


void deleteAtFront(struct Node **head) {
    if (*head == NULL) return;
    struct Node *last = (*head)->prev;
    struct Node *temp = *head;

    if (*head == (*head)->next) {
        *head = NULL;
    } else {
        *head = (*head)->next;
        (*head)->prev = last;
        last->next = *head;
    }
    free(temp);
}

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

    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 4);
    insertAtEnd(&head, 5);

    printf("Original list:\n");
    display(head);

    deleteAtFront(&head);
    printf("After deleting at front:\n");
    display(head);
    return 0;
```

}

Q14. WAP TO IMPLEMENT CIRCULAR DOUBLY LINKED LIST. USING INSERTING NEW NODE AT END CREATE LINKED LIST 1->2->3->4->5. USING TRAVERSING DISPLAY THE LIST. USE DELETION AT END ONCE, DISPLAY THE UPDATED LIST.

Ans14: Sample C program:

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

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

void insertAtEnd(struct Node **head, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    if (*head == NULL) {
        newNode->next = newNode;
        newNode->prev = newNode;
        *head = newNode;
        return;
    }
    struct Node *last = (*head)->prev;

    newNode->next = *head;
    (*head)->prev = newNode;
    newNode->prev = last;
    last->next = newNode;
}

void display(struct Node *head) {
    struct Node *temp;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Circular Doubly Linked List: ");
    temp = head;
    do {
        printf("%d", temp->data);
        temp = temp->next;
        if (temp != head) printf("->");
    } while (temp != head);
    printf("\n");
}
void deleteAtEnd(struct Node **head) {
    if (*head == NULL) return;
```

```
    struct Node *last = (*head)->prev;

    if (last == *head) {
        free(last);
        *head = NULL;
    } else {
        struct Node *secondLast = last->prev;
        secondLast->next = *head;
        (*head)->prev = secondLast;
        free(last);
    }
}

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

    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);
    insertAtEnd(&head, 4);
    insertAtEnd(&head, 5);

    printf("Original list:\n");
    display(head);

    deleteAtEnd(&head);
    printf("After deleting at end:\n");
    display(head);

    return 0;
}
```

Q15. WAP TO IMPLEMENT BST. CREATE BST FOR THE DATA 25, 10, 5, 16, 30, 85, 95, 12, 23. USING IN-ORDER TRAVERSAL DISPLAY THE NODES.

Ans15: Sample C program:

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node* createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

```
struct Node* insert(struct Node *root, int value) {
    if (root == NULL) return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}

void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}


int main() {
    int values[] = {25, 10, 5, 16, 30, 85, 95, 12, 23};
    int n = 9, i;
    struct Node *root = NULL;

    for (i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
    return 0;
}
```

Q16. WAP TO IMPLEMENT BST. CREATE BST FOR THE DATA 25, 10, 5, 16, 30, 85, 95, 12, 23. USING PRE-ORDER TRAVERSAL DISPLAY THE NODES.

Ans16: Sample C program:

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node* createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node *root, int value) {
    if (root == NULL) return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}

void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}


int main() {
```

```
    int values[] = {25, 10, 5, 16, 30, 85, 95, 12, 23};
    int n = 9, i;
    struct Node *root = NULL;

    for (i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    printf("Preorder traversal: ");
    preorder(root);
    printf("\n");
    return 0;
}
```

Q17. WAP TO IMPLEMENT BST. CREATE BST FOR THE DATA 25, 10, 5, 16, 30, 85, 95, 12, 23. USING POST-ORDER TRAVERSAL DISPLAY THE NODES.

Ans17: Sample C program:

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node* createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node *root, int value) {
    if (root == NULL) return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}

void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
void preorder(struct Node *root) {
```

```
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}


int main() {
    int values[] = {25, 10, 5, 16, 30, 85, 95, 12, 23};
    int n = 9, i;
    struct Node *root = NULL;

    for (i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    printf("Postorder traversal: ");
    postorder(root);
    printf("\n");
    return 0;
}
```

Q18. WAP TO IMPLEMENT BST. CREATE BST FOR THE DATA 25, 10, 5, 16, 30, 85, 95, 12, 23. USING LEVEL-ORDER TRAVERSAL DISPLAY THE NODES.

Ans18: Sample C program:

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node* createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node *root, int value) {
```

```
    if (root == NULL) return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}

void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}


void levelOrder(struct Node *root) {
    if (root == NULL) return;
    struct Node *queue[100];
    int front = 0, rear = 0;

    queue[rear++] = root;
    while (front < rear) {
        struct Node *current = queue[front++];
        printf("%d ", current->data);
        if (current->left != NULL)
            queue[rear++] = current->left;
        if (current->right != NULL)
            queue[rear++] = current->right;
    }
}

int main() {
    int values[] = {25, 10, 5, 16, 30, 85, 95, 12, 23};
    int n = 9, i;
    struct Node *root = NULL;
```

```
    for (i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    printf("Level-order traversal: ");
    levelOrder(root);
    printf("\n");
    return 0;
}
```

Q19. WAP TO IMPLEMENT BST. CREATE BST FOR THE DATA 25, 10, 5, 16, 30, 85, 95, 12, 23. SEARCH IF 109 EXISTS IN THE BST.

Ans19: Sample C program:

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node* createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node *root, int value) {
    if (root == NULL) return createNode(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}

void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node *root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
```

```
        preorder(root->right);
    }
}

void postorder(struct Node *root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}


struct Node* search(struct Node *root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}

int main() {
    int values[] = {25, 10, 5, 16, 30, 85, 95, 12, 23};
    int n = 9, i;
    struct Node *root = NULL;
    struct Node *result;
    int key = 109;

    for (i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    result = search(root, key);
    if (result == NULL)
        printf("%d not found in BST\n", key);
    else
        printf("%d found in BST\n", key);

    return 0;
}
```

Q20. WAP TO IMPLEMENT MERGE SORT ALGORITHM. IMPLEMENT THE CODE ON THE ARRAY: 10, 2, 5, 1, 4, 6, 3, 7, 9, 8.

Ans20: Sample C program:

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int i = left, j = mid + 1, k = 0;
    int temp[right - left + 1];
```

```
    while (i <= mid && j <= right) {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    }
    while (i <= mid)
        temp[k++] = arr[i++];
    while (j <= right)
        temp[k++] = arr[j++];

    for (i = left, k = 0; i <= right; i++, k++)
        arr[i] = temp[k];
}

void mergeSort(int arr[], int left, int right) {
    int mid;
    if (left < right) {
        mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

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

int main() {
    int arr[] = {10, 2, 5, 1, 4, 6, 3, 7, 9, 8};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array:\n");
    printArray(arr, n);

    mergeSort(arr, 0, n - 1);

    printf("Sorted array using merge sort:\n");
    printArray(arr, n);

    return 0;
}
```

Q21. WAP TO IMPLEMENT QUICK SORT ALGORITHM. IMPLEMENT THE CODE ON THE ARRAY: 10, 2, 5, 1, 4, 6, 3, 7, 9, 8.

Ans21: Sample C program:

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    int j;
    for (j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high) {
    int pi;
    if (low < high) {
        pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

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

int main() {
    int arr[] = {10, 2, 5, 1, 4, 6, 3, 7, 9, 8};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array:\n");
    printArray(arr, n);

    quickSort(arr, 0, n - 1);

    printf("Sorted array using quick sort:\n");
    printArray(arr, n);

    return 0;
}
```