1)Write a program to implement recursion for

(a) Decimal to Binary Conversion using recursion
(b) Write a program to implement recursion for printing pyramid pattern
(c) Write a program to solve Tower of Hanoi Problem with "n' disks using recursion.

2)Write a program to implement arrays and their operations.

3) (a) Write a program to implement Stack and its operations using arrays.

(b) Write a Program to Convert Infix to Postfix Conversion using Stack

(c) Write a Program to implement Postfix Evaluation using Stack

4) a) Write a program to implement Queue and its operations using arrays.

b) Write a program to implement First Come First Serve CPU Scheduling Algorithm.

5) Write a program to implement Singly Linked List and its operations.

6) Write a program to implement Doubly Linked List and its operations.

7) Write a program to implement Circular Singly Linked List and its operations.

8) Write a program to implement Circular Doubly Linked List and its operations.

9) a) Write a program to implement binary search tree creation and traversals.

b) Write a program to implement binary search tree -Searching and deletion.

10) a) Write a program to implement Linear Search

b) Write a program to implement Binary Search

11) Write a program to implement insertion sort

12) Write a program to implement hash table using Linear Probing

13)Write a program to implement Stack and its operations using linked list.

14)Write a program to implement Queue and its operations using linked list.

1a) Decimal to Binary Conversion using recursion

```
#include <stdio.h>
void decimalToBinary(int n) {
    if (n == 0) return;
    decimalToBinary(n / 2);
    printf("%d", n % 2);
}
int main() {
    int num;
    printf("Enter a decimal number: ");
    scanf("%d", &num);
    if (num == 0) printf("0");
    else decimalToBinary(num);
    printf("\n");
    return 0;
}
```

1b) Write a program to implement recursion for printing pyramid pattern

```
#include <stdio.h>
void printSpaces(int space) {
    if (space == 0) return;
    printf(" ");
    printSpaces(space - 1);
}
void printStars(int stars) {
    if (stars == 0) return;
    printf("* ");
    printStars(stars - 1);
}
void pyramid(int current, int total) {
    if (current > total) return;
    printSpaces(total - current);
    printStars(current);
    printf("\n");
    pyramid(current + 1, total);
}
int main() {
    int rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    pyramid(1, rows);
    return 0;
}
```

1c) Write a program to solve Tower of Hanoi Problem with "n' disks using recursion.

```
#include <stdio.h>
void towerOfHanoi(int n, char from, char aux, char to) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", from, to);
        return;
    }
    towerOfHanoi(n - 1, from, to, aux);
    printf("Move disk %d from %c to %c\n", n, from, to);
    towerOfHanoi(n - 1, aux, from, to);
}
int main() {
    int n;
    printf("Enter number of disks: ");
    scanf("%d", &n);
    towerOfHanoi(n, 'A', 'B', 'C');
    return 0;
}
```

2)Write a program to implement arrays and their operations.

```
#include <stdio.h>
#define SIZE 100
int arr[SIZE], n = 0;
void traverse() {
    printf("Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
void insert(int pos, int value) {
    if (n >= SIZE || pos < 0 || pos > n) {
        printf("Insertion not possible at position %d\n", pos);
        return;
    }
    for (int i = n; i > pos; i--) {
        arr[i] = arr[i - 1];
    }
    arr[pos] = value;
    n++;
    printf("Element %d inserted at position %d\n", value, pos);
}
void deleteByValue(int value) {
    int found = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] == value) {
            for (int j = i; j < n - 1; j++) {
                arr[j] = arr[j + 1];
            }
            n--;
            found = 1;
            printf("Element %d deleted\n", value);
            break;
        }
    }
    if (!found) printf("Element %d not found\n", value);
}
void update(int pos, int value) {
    if (pos < 0 || pos >= n) {
        printf("Update not possible at position %d\n", pos);
    } else {
        arr[pos] = value;
        printf("Element at position %d updated to %d\n", pos, value);
    }
}

int main() {
    insert(0, 10);
    insert(1, 20);
    insert(2, 30);
    traverse();
    update(1, 25);
    traverse();
    deleteByValue(10);
    traverse();
    return 0;
```

```
}
```

3) (a) Write a program to implement Stack and its operations using arrays.

```
#include <stdio.h>
#define SIZE 100

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

void push(int val) {
    if (top == SIZE - 1)
        printf("Stack Overflow\n");
    else {
        stack[++top] = val;
        printf("Element %d pushed\n", val);
    }
}

void pop() {
    if (top == -1)
        printf("Stack Underflow\n");
    else
        printf("Element %d popped\n", stack[top--]);
}

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

int main() {
    push(10);
    push(20);
    push(30);
    display();
    pop();
    display();
    return 0;
}
```

(b) Write a Program to Convert Infix to Postfix Conversion using Stack

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

#define SIZE 100
char stack[SIZE];
int top = -1;

void push(char c) { stack[++top] = c; }
char pop() { return stack[top--]; }
char peek() { return stack[top]; }
int precedence(char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0;
}

void infixToPostfix(char *infix) {
    char postfix[SIZE];
    int j = 0;

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

        if (isalnum(c)) {
            postfix[j++] = c;
        } else if (c == '(') {
            push(c);
        } else if (c == ')') {
            while (top != -1 && peek() != '(')
                postfix[j++] = pop();
            pop(); // pop '('
        } else {
            while (top != -1 && precedence(peek()) >= precedence(c))
                postfix[j++] = pop();
            push(c);
        }
    }

    while (top != -1)
        postfix[j++] = pop();
    postfix[j] = '\0';
    printf("Postfix: %s\n", postfix);
}

int main() {
    char expr[SIZE];
    printf("Enter infix expression: ");
    scanf("%s", expr);
    infixToPostfix(expr);
    return 0;
}
```

(c) Write a Program to implement Postfix Evaluation using Stack

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

#define SIZE 100
int stack[SIZE], top = -1;

void push(int val) { stack[++top] = val; }
int pop() { return stack[top--]; }

int main() {
    char expr[SIZE];
    printf("Enter postfix expression: ");
    scanf("%s", expr);

    for (int i = 0; expr[i] != '\0'; i++) {
        char c = expr[i];
        if (isdigit(c)) {
            push(c - '0');
        } else {
            int b = pop();
            int a = pop();
            switch (c) {
                case '+': push(a + b); break;
                case '-': push(a - b); break;
                case '*': push(a * b); break;
                case '/': push(a / b); break;
            }
        }
    }

    printf("Result = %d\n", pop());
    return 0;
}
```

4) a) Write a program to implement Queue and its operations using arrays.

```
#include <stdio.h>
#define SIZE 100

int queue[SIZE], front = -1, rear = -1;

void enqueue(int val) {
    if (rear == SIZE - 1)
        printf("Queue Overflow\n");
    else {
        if (front == -1) front = 0;
        queue[++rear] = val;
        printf("Element %d enqueued\n", val);
    }
}

void dequeue() {
    if (front == -1 || front > rear)
        printf("Queue Underflow\n");
    else
        printf("Element %d dequeued\n", queue[front++]);
}

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

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    display();
    return 0;
}
```

b) Write a program to implement First Come First Serve CPU Scheduling Algorithm.

```
#include <stdio.h>

int main() {
    int n, i;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int bt[n], wt[n], tat[n];

    printf("Enter burst times:\n");
    for (i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &bt[i]);
    }

    wt[0] = 0;
    for (i = 1; i < n; i++)
        wt[i] = wt[i - 1] + bt[i - 1];

    for (i = 0; i < n; i++)
        tat[i] = wt[i] + bt[i];

    printf("\nProcess\tBT\tWT\tTAT\n");
    for (i = 0; i < n; i++)
```

```
        printf("P%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);

    return 0;
}
```

5) Write a program to implement Singly Linked List and its operations

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

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

struct Node* head = NULL;

void insertBeg(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = head;
    head = newNode;
    printf("Inserted %d at beginning\n", val);
}

void insertEnd(int val) {
    if (head == NULL) {
        insertBeg(val);
        return;
    }
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;
    struct Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
    printf("Inserted %d at end\n", val);
}

void insertPos(int pos, int val) {
    if (head == NULL || pos <= 1) {
        insertBeg(val);
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i < pos - 1 && temp->next; i++)
        temp = temp->next;
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = temp->next;
    temp->next = newNode;
    printf("Inserted %d at position %d\n", val, pos);
}

void deleteValue(int val) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node *temp = head, *prev = NULL;
    while (temp && temp->data != val) {
        prev = temp;
        temp = temp->next;
    }
    if (!temp) {
        printf("Value %d not found\n", val);
        return;
    }
    if (!prev) head = head->next;
    else prev->next = temp->next;
    free(temp);
    printf("Deleted %d\n", val);
}

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

int main() {
    insertEnd(10);         // Should call insertBeg
    insertEnd(20);
    insertPos(2, 15);
    insertPos(1, 5);       // Should go to beginning
    insertPos(100, 25);    // Should go to end
    display();
    deleteValue(15);
    deleteValue(99);       // Not found
    deleteValue(10);
    deleteValue(20);
    deleteValue(25);
    deleteValue(5);
    deleteValue(1);        // List is empty now
    display();
    return 0;
}
```

6)Write a program to implement Doubly Linked List and its operations.

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

struct Node {
    int data;
    struct Node *prev, *next;
};
struct Node* head = NULL;

void insertBeg(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->prev = NULL;
    newNode->next = head;
    if (head) head->prev = newNode;
    head = newNode;
    printf("Inserted %d at beginning\n", val);
}

void insertEnd(int val) {
    if (!head) {
        insertBeg(val);
        return;
    }
    struct Node* temp = head;
    while (temp->next) temp = temp->next;
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;
    newNode->prev = temp;
    temp->next = newNode;
    printf("Inserted %d at end\n", val);
}

void insertPos(int pos, int val) {
    if (!head || pos <= 1) {
        insertBeg(val);
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i < pos - 1 && temp->next; i++)
```

```
        temp = temp->next;
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next) temp->next->prev = newNode;
    temp->next = newNode;
    printf("Inserted %d at position %d\n", val, pos);
}

void deleteValue(int val) {
    if (!head) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp && temp->data != val)
        temp = temp->next;
    if (!temp) {
        printf("Value %d not found\n", val);
        return;
    }
    if (temp->prev) temp->prev->next = temp->next;
    else head = temp->next;
    if (temp->next) temp->next->prev = temp->prev;
    free(temp);
    printf("Deleted %d\n", val);
}
```

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

int main() {
    insertEnd(10);
    insertEnd(20);
    insertPos(2, 15);
    insertBeg(5);
    display();
    deleteValue(15);
    deleteValue(100);
    display();
    return 0;
}
```

7) Write a program to implement Circular Singly Linked List and its operations.

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

```
struct Node {
    int data;
    struct Node* next;
};
struct Node* head = NULL;

void insertBeg(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    if (!head) {
        newNode->next = newNode;
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != head) temp = temp->next;
        newNode->next = head;
        temp->next = newNode;
        head = newNode;
    }
    printf("Inserted %d at beginning\n", val);
}

void insertEnd(int val) {
    if (!head) {
        insertBeg(val);
        return;
    }
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    struct Node* temp = head;
    while (temp->next != head) temp = temp->next;
    temp->next = newNode;
    newNode->next = head;
    printf("Inserted %d at end\n", val);
}

void insertPos(int pos, int val) {
    if (!head || pos <= 1) {
        insertBeg(val);
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i < pos - 1 && temp->next != head; i++)
        temp = temp->next;
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = temp->next;
    temp->next = newNode;
    printf("Inserted %d at position %d\n", val, pos);
}

void deleteValue(int val) {
    if (!head) {
        printf("List is empty\n");
        return;
    }
    struct Node *curr = head, *prev = NULL;
    do {
        if (curr->data == val) break;
        prev = curr;
        curr = curr->next;
    } while (curr != head);

    if (curr->data != val) {
        printf("Value %d not found\n", val);
        return;
    }

    if (curr == head && curr->next == head) {
        head = NULL;
    } else if (curr == head) {
        struct Node* temp = head;
        while (temp->next != head) temp = temp->next;
        head = head->next;
        temp->next = head;
    } else {
        prev->next = curr->next;
    }
    free(curr);
    printf("Deleted %d\n", val);
}

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

int main() {
    insertEnd(10);
    insertEnd(20);
    insertPos(2, 15);
    insertBeg(5);
    display();
    deleteValue(15);
    deleteValue(99);
    display();
    return 0;
}
```

8) Write a program to implement Circular Doubly Linked List and its operations.

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

struct Node {
    int data;
    struct Node *prev, *next;
};
struct Node* head = NULL;
```

```
void insertBeg(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    if (!head) {
        newNode->next = newNode->prev = newNode;
        head = newNode;
    } else {
        struct Node* last = head->prev;
        newNode->next = head;
        newNode->prev = last;
        last->next = head->prev = newNode;
        head = newNode;
    }
    printf("Inserted %d at beginning\n", val);
}

void insertEnd(int val) {
    if (!head) {
        insertBeg(val);
        return;
    }
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    struct Node* last = head->prev;
    newNode->next = head;
    newNode->prev = last;
    last->next = head->prev = newNode;
    printf("Inserted %d at end\n", val);
```

```
}

void insertPos(int pos, int val) {
    if (!head || pos <= 1) {
        insertBeg(val);
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i < pos - 1 && temp->next != head; i++)
        temp = temp->next;
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = temp->next;
    newNode->prev = temp;
    temp->next->prev = newNode;
    temp->next = newNode;
    printf("Inserted %d at position %d\n", val, pos);
}

void deleteValue(int val) {
    if (!head) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    do {
        if (temp->data == val) break;
        temp = temp->next;
    } while (temp != head);
    if (temp->data != val) {
        printf("Value %d not found\n", val);
        return;
    }

    if (temp->next == temp) {
        head = NULL;
    } else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
        if (temp == head)
            head = temp->next;
    }
    free(temp);
    printf("Deleted %d\n", val);
}

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

int main() {
    insertEnd(10);
    insertBeg(5);
    insertPos(2, 7);
    insertEnd(15);
    display();
    deleteValue(7);
    deleteValue(99);
    display();
    return 0;
}
```

9) a) Write a program to implement binary search tree creation and traversals.

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

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

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

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

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

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

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

int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);
    printf("Inorder: "); inorder(root); printf("\n");
    printf("Preorder: "); preorder(root); printf("\n");
    printf("Postorder: "); postorder(root); printf("\n");
    return 0;
}
```

b) Write a program to implement binary search tree -Searching and deletion.

```
struct Node* search(struct Node* root, int val) {
    if (!root || root->data == val) return root;
    if (val < root->data) return search(root->left, val);
    return search(root->right, val);
}

struct Node* findMin(struct Node* node) {
    while (node->left) node = node->left;
    return node;
}
```

```
struct Node* delete(struct Node* root, int val) {
    if (!root) return NULL;
    if (val < root->data) root->left = delete(root->left, val);
    else if (val > root->data) root->right = delete(root->right, val);
    else {
        if (!root->left) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        }
        if (!root->right) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = delete(root->right, temp->data);
    }
    return root;
}
```

10) a) Write a program to implement Linear Search

```
#include <stdio.h>

int linearSearch(int a[], int n, int key) {
    for (int i = 0; i < n; i++)
        if (a[i] == key) return i;
    return -1;
}
```

```
int main() {
    int a[] = {4, 2, 7, 1, 9}, key = 7;
    int index = linearSearch(a, 5, key);
    if (index != -1) printf("Found at index %d\n", index);
    else printf("Not found\n");
    return 0;
}
```

b) Write a program to implement Binary Search

```
#include <stdio.h>

int binarySearch(int a[], int n, int key) {
    int l = 0, r = n - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (a[m] == key) return m;
        if (a[m] < key) l = m + 1;
        else r = m - 1;
    }
    return -1;
}

int main() {
    int a[] = {1, 3, 5, 7, 9}, key = 7;
    int index = binarySearch(a, 5, key);
    if (index != -1) printf("Found at index %d\n", index);
    else printf("Not found\n");
    return 0;
}
```

11) Write a program to implement insertion sort

```
#include <stdio.h>


void insertionSort(int a[], int n) {
    for (int i = 1; i < n; i++) {
        int key = a[i], j = i - 1;
        while (j >= 0 && a[j] > key)
            a[j + 1] = a[j--];
        a[j + 1] = key;
    }
}


int main() {
    int a[] = {5, 3, 8, 1, 2}, n = 5;
    insertionSort(a, n);
    printf("Sorted: ");
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    return 0;
}
```

12) Write a program to implement hash table using Linear Probing

```
#include <stdio.h>
#define SIZE 10


int hashTable[SIZE];


void insert(int val) {
    int i, idx = val % SIZE;
    for (i = 0; i < SIZE; i++) {
        int try = (idx + i) % SIZE;
        if (hashTable[try] == 0) {
            hashTable[try] = val;
            printf("Inserted %d at index %d\n", val, try);
            return;
        }
    }
    printf("Hash table full!\n");
}

void display() {
    for (int i = 0; i < SIZE; i++)
        printf("%d: %d\n", i, hashTable[i]);
}

int main() {
    insert(10);
    insert(20);
    insert(30);
    insert(25);
    display();
    return 0;
}
```

13)Write a program to implement Stack and its operations using linked list.

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

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

struct Node* top = NULL;

void push(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = top;
    top = newNode;
    printf("Pushed %d\n", val);
}

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

void display() {
    struct Node* temp = top;
    printf("Stack: ");
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    push(10); push(20); pop(); display(); return 0;
}
```

14)Write a program to implement Queue and its operations using linked list.

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

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

struct Node *front = NULL, *rear = NULL;

void enqueue(int val) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;
    if (!rear) front = rear = newNode;
    else {
        rear->next = newNode;
        rear = newNode;
    }
    printf("Enqueued %d\n", val);
}

void dequeue() {
    if (!front) {
        printf("Queue is empty\n");
        return;
    }
    struct Node* temp = front;
    front = front->next;
    if (!front) rear = NULL;
    printf("Dequeued %d\n", temp->data);
    free(temp);
}

void display() {
    struct Node* temp = front;
    printf("Queue: ");
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    enqueue(10); enqueue(20); dequeue(); display(); return 0;
}
```