2.

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

#define MAX 100

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

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

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

void infixToPostfix(char infix[], char postfix[]) {
    int i, j = 0;
    char c;
    for (i = 0; infix[i]; i++) {
        c = infix[i];

        if (isalnum(c))
            postfix[j++] = c;

        else if (c == '(')
            push(c);

        else if (c == ')') {
            while (stack[top] != '(')
                postfix[j++] = pop();
            pop();
        }

        else {
            while (top != -1 && precedence(stack[top]) >=
precedence(c))
                postfix[j++] = pop();
            push(c);
        }
    }
    while (top != -1)
        postfix[j++] = pop();

    postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];

    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("%s", postfix);

    return 0;
}
```

3.a

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

#define MAX 50
```

```
int stack[MAX], top = -1;

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

int evaluatePostfix(char exp[]) {
    int i, a, b;

    for (i = 0; exp[i]; i++) {
        if (isdigit(exp[i]))
            push(exp[i] - '0');
        else {
            b = pop();
            a = pop();

            if (exp[i] == '+') push(a + b);
            else if (exp[i] == '-') push(a - b);
            else if (exp[i] == '*') push(a * b);
            else if (exp[i] == '/') push(a / b);
            else if (exp[i] == '%') push(a % b);
            else if (exp[i] == '^') push(pow(a, b));
        }
    }
    return pop();
}

int main() {
    char exp[MAX];
    scanf("%s", exp);
    printf("%d", evaluatePostfix(exp));
    return 0;
}
```

b.

```
#include <stdio.h>

void hanoi(int n, char A, char B, char C) {
    if (n == 0) return;

    hanoi(n - 1, A, C, B);
    printf("Move disk %d from %c --> %c\n", n, A, C);
    hanoi(n - 1, B, A, C);
}
```

```
int main() {
    int n;
    scanf("%d", &n);
    hanoi(n, 'A', 'B', 'C');
    return 0;
}
```

4.

```
#include <stdio.h>

#define MAX 5
char q[MAX];
int f = -1, r = -1;

int full()  { return (r + 1) % MAX == f; }
int empty() { return f == -1; }

void insert() {
    char x;
    if (full()) { printf("Overflow\n"); return; }
    scanf(" %c", &x);
    if (empty()) f = 0;
    r = (r + 1) % MAX;
    q[r] = x;
}

void delete() {
    if (empty()) { printf("Underflow\n"); return; }
    printf("Deleted %c\n", q[f]);
    if (f == r) f = r = -1;
    else f = (f + 1) % MAX;
}

void display() {
    if (empty()) { printf("Empty\n"); return; }
    int i = f;
    while (1) {
        printf("%c ", q[i]);
        if (i == r) break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}

void demo() {
    f = r = -1;
    delete();                // Underflow
    for (int i = 0; i < MAX; i++) insert('A' + i);
    insert('Z');             // Overflow
}

int main() {
    int ch;
    do {
        printf("\n1.Insert 2.Delete 3.Display 4.Demo 5.Exit\n");
        scanf("%d", &ch);
        if (ch == 1) insert();
        else if (ch == 2) delete();
        else if (ch == 3) display();
        else if (ch == 4) demo();
    } while (ch != 5);
    return 0;
}
```

5.

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

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

struct Node *head = NULL;

// Create node
```

```
struct Node* createNode(int data) {
    struct Node *n = (struct Node*)malloc(sizeof(struct Node));
    n->data = data;
    n->next = NULL;
    return n;
}

// Insert at beginning
void insertBeg(int data) {
    struct Node *n = createNode(data);
    n->next = head;
    head = n;
}

// Insert at end
void insertEnd(int data) {
    struct Node *n = createNode(data);

    if (head == NULL) {
        head = n;
        return;
    }
    struct Node *t = head;
    while (t->next != NULL)
        t = t->next;

    t->next = n;
}

// Insert at position
void insertPos(int data, int pos) {
    if (pos == 1) {
        insertBeg(data);
        return;
    }

    struct Node *t = head;
    for (int i = 1; i < pos - 1 && t != NULL; i++)
        t = t->next;

    if (t == NULL) {
        printf("Invalid position\n");
        return;
    }

    struct Node *n = createNode(data);
    n->next = t->next;
    t->next = n;
}

// Delete from beginning
void deleteBeg() {
    if (head == NULL) {
        printf("List empty\n");
        return;
    }

    struct Node *t = head;
    head = head->next;
    free(t);
}

// Delete from end
void deleteEnd() {
    if (head == NULL) {
        printf("List empty\n");
        return;
    }

    if (head->next == NULL) {
        free(head);
        head = NULL;
        return;
    }

    struct Node *t = head;
    while (t->next->next != NULL)
        t = t->next;

    free(t->next);
    t->next = NULL;
}

// Delete from position
void deletePos(int pos) {
    if (head == NULL) {
        printf("List empty\n");
        return;
    }

    if (pos == 1) {
        deleteBeg();
        return;
    }

    struct Node *t = head;
    for (int i = 1; i < pos - 1 && t != NULL; i++)
        t = t->next;

    if (t == NULL || t->next == NULL) {
        printf("Invalid position\n");
        return;
    }

    struct Node *del = t->next;
    t->next = del->next;
    free(del);
}
```

```
// Display list
void display() {
    struct Node *t = head;

    if (t == NULL) {
        printf("List empty\n");
        return;
    }

    while (t != NULL) {
        printf("%d -> ", t->data);
        t = t->next;
    }
    printf("NULL\n");
}

// Main
int main() {
    int choice, data, pos;

    while (1) {
        printf("\n1.Insert Beg");
printf("\n2.Insert End");
printf("\n3.Insert Pos");
printf("\n4.Delete Beg");
printf("\n5.Delete End");
printf("\n6.Delete Pos");
printf("\n7.Display");
printf("\n8.Exit");
printf("\nEnter choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        scanf("%d", &data);
        insertBeg(data);
        break;
    case 2:
        scanf("%d", &data);
        insertEnd(data);
        break;
    case 3:
        scanf("%d %d", &data, &pos);
        insertPos(data, pos);
```

```
                break;
            case 4:
                deleteBeg();
                break;
            case 5:
                deleteEnd();
                break;
            case 6:
                scanf("%d", &pos);
                deletePos(pos);
                break;
            case 7:
                display();
                break;
            case 8:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
}    }
}
```

8.a

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

/* Structure for BST Node */
struct node {
    int data;
    struct node *left, *right;
};

/* Function to create a new node */
struct node* createNode(int data) {
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
```

```
/* Function to insert a node into BST */
struct node* insert(struct node* root, int data) {
    if (root == NULL)
        return createNode(data);

    if (data < root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data); // duplicates go to right

    return root;
}

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

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

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

int main() {
    struct node* root = NULL;
    int choice, ch;
    int arr[] = {6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    int created = 0;

    do {
        printf("\n--- Binary Search Tree Menu ---\n");
        printf("1. Create BST\n");
        printf("2. Traverse BST\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {

        case 1:
            if (created) {
                printf("\nBST already created.\n");
                break;
            }
            for (int i = 0; i < n; i++)
                root = insert(root, arr[i]);
            created = 1;
        printf("\nBST created successfully.\n");
        break;

    case 2:
        if (!created) {
            printf("\nCreate the BST first.\n");
            break;
        }
        printf("\n1. Inorder Traversal\n");
        printf("2. Preorder Traversal\n");
        printf("3. Postorder Traversal\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
        case 1:
            printf("\nInorder Traversal: ");
            inorder(root);
            break;
        case 2:
            printf("\nPreorder Traversal: ");
            preorder(root);
```

```
                break;
            case 3:
                printf("\nPostorder Traversal: ");
                postorder(root);
                break;
            default:
                printf("\nInvalid traversal choice\n");
            }
            printf("\n");
            break;

        case 3:
            printf("\nExiting program...\n");
            break;

        default:
            printf("\nInvalid choice. Try again.\n");
        }

    } while (choice != 3);

    return 0;
```

```
}
```

9.

```
#include <stdio.h>

int adj[10][10], visited[10], n;
int queue[10], front = -1, rear = -1;

/* Enqueue function */
void enqueue(int v) {
    queue[++rear] = v;
}

/* Dequeue function */
int dequeue() {
    return queue[++front];
}

/* BFS Traversal */
void bfs(int start) {
    int i, v;
    visited[start] = 1;
    enqueue(start);
    while (front != rear) {
        v = dequeue();
        printf("%d ", v);

        for (i = 0; i < n; i++) {
            if (adj[v][i] == 1 && visited[i] == 0) {
                visited[i] = 1;
                enqueue(i);
            }
        }
    }
}

int main() {
    int i, j, start;

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
```

```
        for (j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);

    for (i = 0; i < n; i++)
        visited[i] = 0;

    printf("Enter starting vertex (0 to %d): ", n - 1);
    scanf("%d", &start);

    printf("\nBFS Traversal: ");
    bfs(start);

    return 0;
}
```

9.b

```
#include <stdio.h>

int adj[10][10], visited[10], n;
```

```
/* DFS function */
void dfs(int v) {
    int i;
    visited[v] = 1;

    for (i = 0; i < n; i++) {
        if (adj[v][i] == 1 && visited[i] == 0) {
            dfs(i);
        }
    }
}

int main() {
    int i, j, flag = 1;

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);
    for (i = 0; i < n; i++)
        visited[i] = 0;

    /* Perform DFS from vertex 0 */
    dfs(0);

    /* Check if all vertices are visited */
    for (i = 0; i < n; i++) {
        if (visited[i] == 0) {
            flag = 0;
            break;
        }
    }

    if (flag)
        printf("\nGraph is CONNECTED\n");
    else
        printf("\nGraph is NOT CONNECTED\n");

    return 0;
}
```

10.

```
#include <stdio.h>

#define MAX 100

int hashTable[MAX];
int m;
int hashFunction(int key)
{
    return key % m;
}
void insert(int key)
{
    int index = hashFunction(key);

    while (hashTable[index] != -1)
    {
        index = (index + 1) % m;
    }

    hashTable[index] = key;
```

```c
}

/* Display hash table */
void display()
{
    int i;
    printf("\nHash Table:\n");
    for (i = 0; i < m; i++)
    {
        if (hashTable[i] != -1)
            printf("Address %02d : %d\n", i, hashTable[i]);
        else
            printf("Address %02d : Empty\n", i);
    }
}

int main()
{
    int n, key, i;

    printf("Enter size of hash table (m): ");
    scanf("%d", &m);
    for (i = 0; i < m; i++)
        hashTable[i] = -1;

    printf("Enter number of employee records (N): ");
    scanf("%d", &n);

    printf("Enter %d employee keys (4-digit):\n", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &key);
        insert(key);
    }

    display();

    return 0;
}
```