Module I: Introduction to Data Structures and Arrays

---

1. Declare and print elements of a 1D array of 5 integers

```
#include <iostream>
using namespace std;

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    for (int i = 0; i < 5; i++) {
        cout << "Element " << i << ": " << arr[i] << endl;
    }
    return 0;
}
```

---

2. Input 10 numbers in an array and display them

```
#include <iostream>
using namespace std;

int main() {
    int arr[10];
    cout << "Enter 10 numbers: ";
    for (int i = 0; i < 10; i++)
        cin >> arr[i];
    cout << "You entered: ";
    for (int i = 0; i < 10; i++)
        cout << arr[i] << " ";
    return 0;
}
```

---

3. Sum of all elements in a 1D array

```
#include <iostream>
using namespace std;

int main() {
    int arr[5] = {2, 4, 6, 8, 10}, sum = 0;
    for (int i = 0; i < 5; i++)
        sum += arr[i];
    cout << "Sum = " << sum;
    return 0;
}
```

---

4. Find maximum and minimum in an array

```
#include <iostream>
using namespace std;

int main() {
    int arr[5] = {11, 22, 3, 44, 5};
    int max = arr[0], min = arr[0];
```

```
    for (int i = 1; i < 5; i++) {
        if (arr[i] > max) max = arr[i];
        if (arr[i] < min) min = arr[i];
    }
    cout << "Max = " << max << ", Min = " << min;
    return 0;
}
```

---

5. Reverse the elements of a 1D array

```
#include <iostream>
using namespace std;

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    cout << "Reversed array: ";
    for (int i = 4; i >= 0; i--)
        cout << arr[i] << " ";
    return 0;
}
```

---

6. Input a 2D array (3x3) and display it

```
#include <iostream>
using namespace std;

int main() {
    int arr[3][3];
    cout << "Enter 3x3 matrix:\n";
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cin >> arr[i][j];

    cout << "Matrix is:\n";
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++)
            cout << arr[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

---

7. Element-wise addition of two 1D arrays

```
#include <iostream>
using namespace std;

int main() {
    int a[5] = {1, 2, 3, 4, 5}, b[5] = {5, 4, 3, 2, 1}, sum[5];
    for (int i = 0; i < 5; i++)
        sum[i] = a[i] + b[i];
    cout << "Sum: ";
    for (int i = 0; i < 5; i++)
        cout << sum[i] << " ";
```

```
    return 0;
}
```

---

8. Count even and odd numbers in an array

```
#include <iostream>
using namespace std;

int main() {
    int arr[6] = {1, 2, 3, 4, 5, 6};
    int even = 0, odd = 0;
    for (int i = 0; i < 6; i++) {
        if (arr[i] % 2 == 0) even++;
        else odd++;
    }
    cout << "Even: " << even << ", Odd: " << odd;
    return 0;
}
```

---

9. Copy elements of one array into another

```
#include <iostream>
using namespace std;

int main() {
    int src[5] = {5, 10, 15, 20, 25}, dest[5];
    for (int i = 0; i < 5; i++)
        dest[i] = src[i];
    cout << "Copied array: ";
    for (int i = 0; i < 5; i++)
        cout << dest[i] << " ";
    return 0;
}
```

---

10. Insert an element at a specific position in a 1D array

```
#include <iostream>
using namespace std;

int main() {
    int arr[6] = {10, 20, 30, 40, 50};
    int pos = 2, val = 25; // insert 25 at index 2
    for (int i = 5; i > pos; i--)
        arr[i] = arr[i - 1];
    arr[pos] = val;
    cout << "Array after insertion: ";
    for (int i = 0; i < 6; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Module II: Stacks and Queues

---

1. Stack using arrays (push and pop)

```
#include <iostream>
using namespace std;

#define SIZE 5
int stack[SIZE], top = -1;

void push(int val) {
   if (top == SIZE - 1) cout << "Stack Overflow\n";
   else stack[++top] = val;
}

void pop() {
   if (top == -1) cout << "Stack Underflow\n";
   else cout << "Popped: " << stack[top--] << endl;
}

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

---

2. Display top element of a stack without removing it

```
#include <iostream>
using namespace std;

int main() {
   int stack[5] = {10, 20, 30}, top = 2;
   cout << "Top element: " << stack[top] << endl;
   return 0;
}
```

---

3. Check if a stack is empty or full

```
#include <iostream>
using namespace std;

#define SIZE 5

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

   if (top == -1)
      cout << "Stack is empty\n";

   top = SIZE - 1;
   if (top == SIZE - 1)
      cout << "Stack is full\n";
```

```
    return 0;
}
```

---

4. Queue using arrays (enqueue and dequeue)

```
#include <iostream>
using namespace std;

#define SIZE 5
int queue[SIZE], front = -1, rear = -1;

void enqueue(int val) {
    if (rear == SIZE - 1) cout << "Queue Full\n";
    else {
        if (front == -1) front = 0;
        queue[++rear] = val;
    }
}

void dequeue() {
    if (front == -1 || front > rear) cout << "Queue Empty\n";
    else cout << "Dequeued: " << queue[front++] << endl;
}

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

---

5. Display front and rear elements of a queue

```
#include <iostream>
using namespace std;

int main() {
    int queue[5] = {10, 20, 30}, front = 0, rear = 2;
    cout << "Front: " << queue[front] << ", Rear: " << queue[rear] << endl;
    return 0;
}
```

---

6. Circular queue operations

```
#include <iostream>
using namespace std;

#define SIZE 5
int cq[SIZE], front = -1, rear = -1;

void enqueue(int val) {
    if ((rear + 1) % SIZE == front) {
        cout << "Queue Full\n";
    } else {
        if (front == -1) front = 0;
```

```
        rear = (rear + 1) % SIZE;
        cq[rear] = val;
    }
}

void dequeue() {
    if (front == -1) {
        cout << "Queue Empty\n";
    } else {
        cout << "Dequeued: " << cq[front] << endl;
        if (front == rear) front = rear = -1;
        else front = (front + 1) % SIZE;
    }
}

int main() {
    enqueue(1); enqueue(2); enqueue(3);
    dequeue(); dequeue();
    return 0;
}
```

---

7. Infix to postfix expression (basic, single-digit operands)

```
#include <iostream>
#include <stack>
using namespace std;

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

string infixToPostfix(string infix) {
    stack<char> s;
    string postfix = "";
    for (char ch : infix) {
        if (isalnum(ch)) postfix += ch;
        else {
            while (!s.empty() && precedence(s.top()) >= precedence(ch)) {
                postfix += s.top(); s.pop();
            }
            s.push(ch);
        }
    }
    while (!s.empty()) {
        postfix += s.top(); s.pop();
    }
    return postfix;
}

int main() {
    string infix = "a+b*c";
    cout << "Postfix: " << infixToPostfix(infix);
    return 0;
}
```

---

8. Evaluate postfix expression (digits only)

```
#include <iostream>
#include <stack>
using namespace std;

int evalPostfix(string exp) {
    stack<int> s;
    for (char ch : exp) {
        if (isdigit(ch)) s.push(ch - '0');
        else {
            int b = s.top(); s.pop();
            int a = s.top(); s.pop();
            switch (ch) {
                case '+': s.push(a + b); break;
                case '-': s.push(a - b); break;
                case '*': s.push(a * b); break;
                case '/': s.push(a / b); break;
            }
        }
    }
    return s.top();
}

int main() {
    string exp = "53+82-*"; // (5+3)*(8-2)
    cout << "Result: " << evalPostfix(exp);
    return 0;
}
```

---

9. Check for balanced parentheses using stack

```
#include <iostream>
#include <stack>
using namespace std;

bool isBalanced(string expr) {
    stack<char> s;
    for (char ch : expr) {
        if (ch == '(') s.push(ch);
        else if (ch == ')') {
            if (s.empty()) return false;
            s.pop();
        }
    }
    return s.empty();
}

int main() {
    string expr = "(a+b)*(c+d)";
    cout << (isBalanced(expr) ? "Balanced" : "Not Balanced");
    return 0;
}
```

---

10. Menu-driven stack operations using array

```
#include <iostream>
using namespace std;

#define SIZE 5
int stack[SIZE], top = -1;

void push() {
   int val;
   if (top == SIZE - 1) cout << "Stack Overflow\n";
   else {
      cout << "Enter value: ";
      cin >> val;
      stack[++top] = val;
   }
}

void pop() {
   if (top == -1) cout << "Stack Underflow\n";
   else cout << "Popped: " << stack[top--] << endl;
}

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

int main() {
   int choice;
   do {
      cout << "\n1.Push\n2.Pop\n3.Display\n4.Exit\nChoice: ";
      cin >> choice;
      switch (choice) {
         case 1: push(); break;
         case 2: pop(); break;
         case 3: display(); break;
      }
   } while (choice != 4);
   return 0;
}
```

Module III: Linked Lists

---

1. Create a singly linked list and display it

```
#include <iostream>
using namespace std;

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

int main() {
   Node* head = new Node{10, nullptr};
   head->next = new Node{20, nullptr};
   head->next->next = new Node{30, nullptr};

   Node* temp = head;
   cout << "Linked List: ";
   while (temp) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   return 0;
}
```

---

2. Insert a node at the beginning

```
#include <iostream>
using namespace std;

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

void insertAtBeginning(Node*& head, int val) {
   Node* newNode = new Node{val, head};
   head = newNode;
}

void display(Node* head) {
   while (head) {
      cout << head->data << " ";
      head = head->next;
   }
}

int main() {
   Node* head = nullptr;
   insertAtBeginning(head, 30);
   insertAtBeginning(head, 20);
   insertAtBeginning(head, 10);
   display(head);
   return 0;
}
```

---

3. Insert a node at the end

```
#include <iostream>
using namespace std;

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

void insertAtEnd(Node*& head, int val) {
    Node* newNode = new Node{val, nullptr};
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) temp = temp->next;
    temp->next = newNode;
}

void display(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
}

int main() {
    Node* head = nullptr;
    insertAtEnd(head, 10);
    insertAtEnd(head, 20);
    insertAtEnd(head, 30);
    display(head);
    return 0;
}
```

---

4. Delete a node from the beginning

```
#include <iostream>
using namespace std;

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

void deleteFromBeginning(Node*& head) {
    if (!head) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}

void display(Node* head) {
    while (head) {
```

```
        cout << head->data << " ";
        head = head->next;
    }
}

int main() {
    Node* head = new Node{10, new Node{20, new Node{30, nullptr}}};
    deleteFromBeginning(head);
    display(head);
    return 0;
}
```

---

5. Delete a node from the end

```
#include <iostream>
using namespace std;

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

void deleteFromEnd(Node*& head) {
    if (!head) return;
    if (!head->next) {
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while (temp->next->next) temp = temp->next;
    delete temp->next;
    temp->next = nullptr;
}

void display(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
}

int main() {
    Node* head = new Node{10, new Node{20, new Node{30, nullptr}}};
    deleteFromEnd(head);
    display(head);
    return 0;
}
```

---

6. Search an element in a linked list

```
#include <iostream>
using namespace std;

struct Node {
    int data;
```

```
    Node* next;
};

bool search(Node* head, int key) {
    while (head) {
        if (head->data == key) return true;
        head = head->next;
    }
    return false;
}

int main() {
    Node* head = new Node{10, new Node{20, new Node{30, nullptr}}};
    cout << (search(head, 20) ? "Found" : "Not Found");
    return 0;
}
```

---

7. Reverse a singly linked list

```
#include <iostream>
using namespace std;

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

Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* curr = head;
    while (curr) {
        Node* next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}

void display(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
}

int main() {
    Node* head = new Node{10, new Node{20, new Node{30, nullptr}}};
    head = reverse(head);
    display(head);
    return 0;
}
```

---

8. Create and display a doubly linked list

```
#include <iostream>
```

```
using namespace std;

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

void display(DNode* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
}

int main() {
    DNode* head = new DNode{10, nullptr, nullptr};
    DNode* second = new DNode{20, head, nullptr};
    head->next = second;
    DNode* third = new DNode{30, second, nullptr};
    second->next = third;

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

---

9. Insert and delete nodes in a circular linked list

```
#include <iostream>
using namespace std;

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

void insert(Node*& tail, int val) {
    Node* newNode = new Node{val, nullptr};
    if (!tail) {
        newNode->next = newNode;
        tail = newNode;
    } else {
        newNode->next = tail->next;
        tail->next = newNode;
        tail = newNode;
    }
}

void deleteNode(Node*& tail, int key) {
    if (!tail) return;
    Node* curr = tail->next, *prev = tail;
    do {
        if (curr->data == key) {
            if (curr == tail) tail = prev;
            if (curr == tail->next) tail->next = curr->next;
            prev->next = curr->next;
            delete curr;
            return;
```

```
        }
        prev = curr;
        curr = curr->next;
    } while (curr != tail->next);
}

void display(Node* tail) {
    if (!tail) return;
    Node* temp = tail->next;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != tail->next);
}

int main() {
    Node* tail = nullptr;
    insert(tail, 10);
    insert(tail, 20);
    insert(tail, 30);
    deleteNode(tail, 20);
    display(tail);
    return 0;
}
```

---

10. Count number of nodes in a linked list

```
#include <iostream>
using namespace std;

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

int count(Node* head) {
    int cnt = 0;
    while (head) {
        cnt++;
        head = head->next;
    }
    return cnt;
}

int main() {
    Node* head = new Node{10, new Node{20, new Node{30, nullptr}}};
    cout << "Count: " << count(head);
    return 0;
}
```

Module IV: Trees

---

1. Create a simple binary tree and display root, left, and right

```
#include <iostream>
using namespace std;

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

int main() {
    Node* root = new Node{1, nullptr, nullptr};
    root->left = new Node{2, nullptr, nullptr};
    root->right = new Node{3, nullptr, nullptr};

    cout << "Root: " << root->data << endl;
    cout << "Left Child: " << root->left->data << endl;
    cout << "Right Child: " << root->right->data << endl;
    return 0;
}
```

---

2. In-order traversal of a binary tree

```
#include <iostream>
using namespace std;

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

void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

int main() {
    Node* root = new Node{1, new Node{2, nullptr, nullptr}, new Node{3, nullptr, nullptr}};
    inorder(root);
    return 0;
}
```

---

3. Pre-order traversal of a binary tree

```
#include <iostream>
using namespace std;

struct Node {
```

```
    int data;
    Node* left, *right;
};

void preorder(Node* root) {
    if (root) {
        cout << root->data << " ";
        preorder(root->left);
        preorder(root->right);
    }
}

int main() {
    Node* root = new Node{1, new Node{2, nullptr, nullptr}, new Node{3, nullptr, nullptr}};
    preorder(root);
    return 0;
}
```

---

4. Post-order traversal of a binary tree

```
#include <iostream>
using namespace std;

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

void postorder(Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        cout << root->data << " ";
    }
}

int main() {
    Node* root = new Node{1, new Node{2, nullptr, nullptr}, new Node{3, nullptr, nullptr}};
    postorder(root);
    return 0;
}
```

---

5. Insert a node in a Binary Search Tree (BST)

```
#include <iostream>
using namespace std;

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

Node* insert(Node* root, int key) {
    if (!root) return new Node{key, nullptr, nullptr};
    if (key < root->data) root->left = insert(root->left, key);
    else root->right = insert(root->right, key);
```

```
    return root;
}

void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    inorder(root);
    return 0;
}
```

---

6. Search for a node in a BST

```
#include <iostream>
using namespace std;

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

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

int main() {
    Node* root = new Node{40, new Node{20, nullptr, nullptr}, new Node{60, nullptr, nullptr}};
    cout << (search(root, 20) ? "Found" : "Not Found");
    return 0;
}
```

---

7. Find the minimum and maximum in a BST

```
#include <iostream>
using namespace std;

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

Node* findMin(Node* root) {
    while (root && root->left) root = root->left;
    return root;
}
```

```
Node* findMax(Node* root) {
    while (root && root->right) root = root->right;
    return root;
}

int main() {
    Node* root = new Node{40, new Node{20, nullptr, nullptr}, new Node{60, nullptr, nullptr}};
    cout << "Min: " << findMin(root)->data << ", Max: " << findMax(root)->data;
    return 0;
}
```

---

8. Delete a node from a BST

```
#include <iostream>
using namespace std;

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

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

Node* deleteNode(Node* root, int key) {
    if (!root) return nullptr;
    if (key < root->data) root->left = deleteNode(root->left, key);
    else if (key > root->data) root->right = deleteNode(root->right, key);
    else {
        if (!root->left) return root->right;
        else if (!root->right) return root->left;
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

int main() {
    Node* root = new Node{50, new Node{30, nullptr, nullptr}, new Node{70, nullptr, nullptr}};
    root = deleteNode(root, 30);
    inorder(root);
    return 0;
}
```

---

9. Find the height of a binary tree

```
#include <iostream>
using namespace std;

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

int height(Node* root) {
    if (!root) return 0;
    int lh = height(root->left);
    int rh = height(root->right);
    return 1 + max(lh, rh);
}

int main() {
    Node* root = new Node{1, new Node{2, nullptr, nullptr}, new Node{3, nullptr, nullptr}};
    cout << "Height: " << height(root);
    return 0;
}
```

---

10. Implement left rotation in an AVL Tree

```
#include <iostream>
using namespace std;

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

Node* leftRotate(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;
    y->left = x;
    x->right = T2;
    return y;
}

void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

int main() {
    Node* root = new Node{10, nullptr, new Node{20, nullptr, nullptr}};
    root = leftRotate(root);
    inorder(root);
    return 0;
}
```

1. Program to represent a graph using an adjacency matrix

```
#include <iostream>
using namespace std;

int main() {
    int n, e;
    cout << "Enter number of vertices: ";
    cin >> n;

    int adj[n][n] = {0};

    cout << "Enter number of edges: ";
    cin >> e;

    cout << "Enter edges (u v):\n";
    for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        adj[u][v] = adj[v][u] = 1; // for undirected graph
    }

    cout << "Adjacency Matrix:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << adj[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

---

2. Program to represent a graph using an adjacency list

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, e;
    cout << "Enter number of vertices: ";
    cin >> n;

    vector<int> adj[n];

    cout << "Enter number of edges: ";
    cin >> e;

    cout << "Enter edges (u v):\n";
    for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    cout << "Adjacency List:\n";
```

```
    for (int i = 0; i < n; i++) {
        cout << i << " -> ";
        for (int x : adj[i])
            cout << x << " ";
        cout << endl;
    }

    return 0;
}
```

---

3. Program to perform BFS traversal

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void BFS(int start, vector<int> adj[], int n) {
    vector<bool> visited(n, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    cout << "BFS Traversal: ";

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int x : adj[node]) {
            if (!visited[x]) {
                visited[x] = true;
                q.push(x);
            }
        }
    }
}

int main() {
    int n, e;
    cin >> n >> e;

    vector<int> adj[n];
    for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    BFS(0, adj, n);

    return 0;
}
```

---

4. Program to perform DFS traversal

```
#include <iostream>
#include <vector>
using namespace std;

void DFS(int node, vector<int> adj[], vector<bool> &vis) {
    vis[node] = true;
    cout << node << " ";

    for (int x : adj[node]) {
        if (!vis[x])
            DFS(x, adj, vis);
    }
}

int main() {
    int n, e;
    cin >> n >> e;

    vector<int> adj[n];
    for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<bool> visited(n, false);

    cout << "DFS Traversal: ";
    DFS(0, adj, visited);

    return 0;
}
```

---

5. Program to implement hashing with linear probing

```
#include <iostream>
using namespace std;

int main() {
    int size, n;
    cout << "Enter hash table size: ";
    cin >> size;

    int hashTable[size];
    for (int i = 0; i < size; i++)
        hashTable[i] = -1;

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        int key;
        cin >> key;

        int index = key % size;
```

```
        while (hashTable[index] != -1)
            index = (index + 1) % size;

        hashTable[index] = key;
    }

    cout << "Hash Table:\n";
    for (int i = 0; i < size; i++)
        cout << i << " -> " << hashTable[i] << endl;

    return 0;
}
```

---

6. Program to implement a basic hash function (modulo method)

```
#include <iostream>
using namespace std;

int main() {
    int key, size;
    cout << "Enter key: ";
    cin >> key;
    cout << "Enter hash table size: ";
    cin >> size;

    int hashValue = key % size;

    cout << "Hash Value = " << hashValue << endl;

    return 0;
}
```

---

7. Program to sort an array using Bubble Sort

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    int a[n];

    for (int i = 0; i < n; i++)
        cin >> a[i];

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1])
                swap(a[j], a[j + 1]);
        }
    }

    cout << "Sorted Array: ";
    for (int x : a)
        cout << x << " ";

    return 0;
}
```

---

8. Program to sort an array using Selection Sort

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    int a[n];

    for (int i = 0; i < n; i++)
        cin >> a[i];

    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (a[j] < a[minIndex])
                minIndex = j;
        }
        swap(a[i], a[minIndex]);
    }

    cout << "Sorted Array: ";
    for (int x : a)
        cout << x << " ";

    return 0;
}
```

---

9. Program to sort an array using Insertion Sort

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    int a[n];

    for (int i = 0; i < n; i++)
        cin >> a[i];

    for (int i = 1; i < n; i++) {
        int key = a[i];
        int j = i - 1;

        while (j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }

    cout << "Sorted Array: ";
    for (int x : a)
        cout << x << " ";

    return 0;
```

```
}
```

---

10. Program to implement Merge Sort

```
#include <iostream>
using namespace std;

void merge(int a[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = a[l + i];
    for (int i = 0; i < n2; i++)
        R[i] = a[m + 1 + i];

    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            a[k++] = L[i++];
        else
            a[k++] = R[j++];
    }

    while (i < n1)
        a[k++] = L[i++];
    while (j < n2)
        a[k++] = R[j++];
}

void mergeSort(int a[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(a, l, m);
        mergeSort(a, m + 1, r);
        merge(a, l, m, r);
    }
}

int main() {
    int n;
    cin >> n;
    int a[n];

    for (int i = 0; i < n; i++)
        cin >> a[i];

    mergeSort(a, 0, n - 1);

    cout << "Sorted Array: ";
    for (int x : a)
        cout << x << " ";

    return 0;
}
```