1.1 Stack

```
#include <iostream>
#include <stack> // Fixed the header for stack
using namespace std;

int main() {
   stack<int> mystack; // Fixed the declaration of the stack
   mystack.push(1);
   mystack.push(2);
   mystack.push(3);

   while (!mystack.empty()) {
      cout << mystack.top() << " "; // Corrected the output stream operator
      mystack.pop();
   }

   return 0; // Fixed the return statement
}
```

1.2 Queue

```
#include <iostream>

#include <queue> // Fixed the header for queue

using namespace std;


int main() {

   queue<int> myqueue; // Fixed the queue declaration


   myqueue.push(1);

   myqueue.push(2);

   myqueue.push(3);


   while (!myqueue.empty()) {

      cout << myqueue.front() << " "; // Corrected the output method

      myqueue.pop(); // Fixed the pop statement

   }


   return 0; // Fixed the return statement

}
```

1.3 List

```
#include <iostream> // Fixed the header
#include <list>
using namespace std;

int main() {
    list<int> mylist; // Fixed the list declaration

    mylist.push_back(1); // Fixed syntax
    mylist.push_back(2); // Fixed syntax
    mylist.push_back(3); // Fixed syntax

    for (const auto& item : mylist) { // Fixed the loop syntax
        cout << item << " "; // Corrected output operator
    }
    cout << endl; // Added a newline for better output formatting
    return 0; // Fixed the return statement
}
```

2 Sum of matrices

```
#include <iostream> // Fixed the include statement
using namespace std;

int main() { // Fixed the main function declaration
    int first[20][20], second[20][20], sum[20][20];
    int rows, cols; // Fixed variable declarations

    cout << "Enter rows and columns: "; // Fixed output statement
    cin >> rows >> cols; // Fixed input statement

    cout << "Enter elements for first matrix: "; // Fixed output statement
    for (int i = 0; i < rows; i++) { // Fixed loop conditions
        for (int j = 0; j < cols; j++) {
            cin >> first[i][j]; // Fixed matrix indexing
        }
```

```
    }

    cout << "Enter elements for second matrix: "; // Fixed output statement
    for (int i = 0; i < rows; i++) { // Fixed loop conditions
        for (int j = 0; j < cols; j++) {
            cin >> second[i][j]; // Fixed matrix indexing
        }
    }

    // Summing the matrices
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            sum[i][j] = first[i][j] + second[i][j]; // Fixed matrix indexing
        }
    }

    cout << "Sum of matrix elements are: " << endl; // Fixed output statement
    for (int i = 0; i < rows; i++) { // Fixed loop conditions
        for (int j = 0; j < cols; j++) {
            cout << sum[i][j] << " "; // Fixed output statement
        }
        cout << endl; // Added newline for better formatting
    }

    return 0; // Fixed the return statement
}
```

3 Stack – ins.,del.,trav.

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

int stack[100], n = 100, top = -1;

void push(int val) {
    if (top >= n - 1) {
```

```
        cout << "Stack Overflow" << endl;
    } else {
        top++; // Increment top before assignment
        stack[top] = val; // Assign value to the new top
    }
}

void pop() {
    if (top <= -1) {
        cout << "Stack Underflow" << endl;
    } else {
        cout << "The popped element is " << stack[top] << endl;
        top--; // Decrement top after popping
    }
}

void display() {
    if (top >= 0) {
        cout << "Stack elements are: ";
        for (int i = top; i >= 0; i--) { // Fixed loop condition
            cout << stack[i] << " "; // Corrected output operator
        }
        cout << endl; // Moved outside the loop
    } else {
        cout << "Stack is empty" << endl; // Fixed error
    }
}

int main() {
    int ch, val;
    cout << "1) Push in stack" << endl;
    cout << "2) Pop from stack" << endl;
    cout << "3) Display stack" << endl;
    cout << "4) Exit" << endl;
```

```
    do {
        cout << "Enter choice: " << endl;
        cin >> ch;
        switch (ch) {
            case 1: {
                cout << "Enter value to be pushed:" << endl;
                cin >> val;
                push(val); // Fixed semicolon
                break;
            }
            case 2: {
                pop();
                break;
            }
            case 3: {
                display();
                break;
            }
            case 4: {
                cout << "Exit" << endl;
                break;
            }
            default: {
                cout << "Invalid Choice" << endl;
            }
        }
    } while (ch != 4);

    return 0;
}
```

4 Queue ins.,del.,trav.

```
#include <iostream> // Fixed the include statement
using namespace std;

int queue[100], n = 100, front = -1, rear = -1;

void Insert() {
    int val;
    if (rear >= n - 1) { // Fixed condition to check for overflow
        cout << "Queue Overflow" << endl;
        return; // Added return statement to exit the function
    }
    if (front == -1) // Fixed to check if the queue is empty
        front = 0;

    cout << "Insert the element in queue: " << endl;
    cin >> val;
    rear++; // Fixed increment of rear
    queue[rear] = val; // Fixed array indexing
}

void Delete() {
    if (front == -1 || front > rear) { // Fixed condition to check for underflow
        cout << "Queue Underflow" << endl;
        return;
    } else {
        cout << "Element deleted from queue is: " << queue[front] << endl; // Fixed array indexing
        front++; // Increment front
    }
}

void Display() {
    if (front == -1) {
        cout << "Queue is empty" << endl;
```

```
    } else {
        cout << "Queue elements are: ";
        for (int i = front; i <= rear; i++) {
            cout << queue[i] << " "; // Fixed output statement
        }
        cout << endl; // Added newline for better formatting
    }
}

int main() {
    int ch;
    cout << "1) Insert element to queue" << endl;
    cout << "2) Delete element from queue" << endl;
    cout << "3) Display all the elements of queue" << endl;
    cout << "4) Exit" << endl;

    do {
        cout << "Enter your choice: " << endl;
        cin >> ch; // Fixed input statement
        switch (ch) {
            case 1:
                Insert();
                break;
            case 2:
                Delete();
                break;
            case 3:
                Display();
                break;
            case 4:
                cout << "Exit" << endl;
                break;
            default:
                cout << "Invalid choice" << endl;
```

```
    }
  } while (ch != 4); // Fixed the loop condition

  return 0; // Fixed the return statement
}
```

5 Priority Queue ins.,del.,trav.

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

class PriorityQueue {
private:
  vector<int> heap;

  void heapifyUp(int index) {
    if (index == 0) return; // Base case, if it's the root, stop
    int parentIndex = (index - 1) / 2;
    // If current node is greater than its parent, swap them
    if (heap[index] > heap[parentIndex]) {
      swap(heap[index], heap[parentIndex]);
      heapifyUp(parentIndex); // Recursively heapify the parent node
    }
  }

  // Function to heapify downwards (used during deletion)
  void heapifyDown(int index) {
    int leftChild = 2 * index + 1;
    int rightChild = 2 * index + 2;
    int largest = index;

    // Find the largest among parent, left child, and right child
    if (leftChild < heap.size() && heap[leftChild] > heap[largest]) {
      largest = leftChild;
    }
```

```
        if (rightChild < heap.size() && heap[rightChild] > heap[largest]) {
            largest = rightChild;
        }

        // If the largest is not the current node, swap and continue heapifying
        if (largest != index) {
            swap(heap[index], heap[largest]);
            heapifyDown(largest);
        }
    }

public:
    // Function to insert an element into the priority queue
    void insert(int value) {
        heap.push_back(value); // Add the new value to the end of the vector
        heapifyUp(heap.size() - 1); // Restore the heap property
    }

    // Function to delete the highest-priority element (root of the heap)
    void deleteMax() {
        if (heap.empty()) {
            cout << "Priority Queue is empty." << endl;
            return;
        }
        // Replace the root with the last element
        heap[0] = heap.back();
        heap.pop_back(); // Remove the last element
        // Restore the heap property
        heapifyDown(0);
    }

    // Function to get the highest-priority element (root of the heap)
    int getMax() {
```

```
        if (heap.empty()) {
            cout << "Priority Queue is empty." << endl;
            return -1;
        }
        return heap[0]; // Return the root element
    }

    // Function to check if the priority queue is empty
    bool isEmpty() {
        return heap.empty();
    }

    // Function to display the elements in the priority queue (heap structure)
    void display() {
        if (heap.empty()) {
            cout << "Priority Queue is empty." << endl;
            return;
        }

        for (int i = 0; i < heap.size(); ++i) {
            cout << heap[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    PriorityQueue pq;
    pq.insert(10);
    pq.insert(30);
    pq.insert(20);
    pq.insert(40);
    pq.insert(50);
    cout << "Priority Queue elements: ";
```

```
    pq.display();
    cout << "Max element: " << pq.getMax() << endl;
    pq.deleteMax();
    cout << "After deletion of max element: ";
    pq.display();
    return 0;
}
```

6 Array ins.,del.,trav.

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

int a[20], n, val, i, pos, choice;

void display();
void insert();
void del();

int main(){
    cout << "\nEnter the size of the array:\t";
    cin >> n;
    cout << "\nEnter the elements for the array:\n";
    for(i = 0; i < n; i++){
        cin >> a[i];
    }
    do {
        cout << "\n\n---------Menu----------\n";
        cout << "1. Insert\n";
        cout << "2. Delete\n";
        cout << "3. Exit\n";
        cout << "-----------------------\n";
        cout << "Enter your choice:\t";
        cin >> choice;
        switch(choice){
            case 1: insert();
```

```
            break;
        case 2: del();
            break;
        case 3: break;
        default: cout << "\nInvalid choice.\n";
      }
   } while(choice != 3);
   return 0;
}

void display(){
   cout << "\nThe array elements are:\n";
   for(i = 0; i < n; i++){
      cout << a[i] << " ";
   }
   cout << endl;
}

void insert(){
   if (n >= 20) {
      cout << "\nArray is full. Cannot insert new element.\n";
      return;
   }
   cout << "\nEnter the position for the new element (1 to " << n + 1 << "):\t";
   cin >> pos;
   if(pos < 1 || pos > n + 1) {
      cout << "\nInvalid position.\n";
      return;
   }
   cout << "Enter the element to be inserted:\t";
   cin >> val;
   for(i = n; i >= pos; i--){
      a[i] = a[i - 1];
   }
```

```
    a[pos - 1] = val;
    n++;
    display();
}

void del(){
    cout << "\nEnter the position of the element to be deleted (1 to " << n << "):\t";
    cin >> pos;
    if(pos < 1 || pos > n) {
        cout << "\nInvalid position.\n";
        return;
    }
    val = a[pos - 1];
    for(i = pos - 1; i < n - 1; i++){
        a[i] = a[i + 1];
    }
    n--;
    cout << "\nThe deleted element is = " << val << endl;
    display();
}
```

7 Implement binary tree ins.,del.,trav.

```
#include <iostream>
#include <stdlib.h>
using namespace std;

void insert(int, int);
void delte(int);
void display(int);
int search(int);
int search1(int, int);

int tree[40], t = 1, x;

int main() {
```

```
int ch, y;
for (int i = 1; i < 40; i++) {
   tree[i] = -1;
}
while (1) {
   cout << "\n1.INSERT\n2.DELETE\n3.DISPLAY\n4.SEARCH\n5.EXIT\nEnter your Choice:\t";
   cin >> ch;
   switch (ch) {
      case 1:
         cout << "Enter the element to Insert\t";
         cin >> x;
         insert(1, x);
         break;
      case 2:
         cout << "Enter the element to Delete\t";
         cin >> x;
         y = search(1);
         if (y != -1)
            delte(y);
         else
            cout << "No Such Element Found in Tree\t";
         break;
      case 3:
         display(1);
         cout << "\n";
         break;
      case 4:
         cout << "Enter the Element to Search:\t";
         cin >> x;
         y = search(1);
         if (y == -1)
            cout << "No such Element Found in Tree\t";
         else
            cout << x << " is in Position " << y;
```

```
            break;
        case 5:
            exit(0);
        default:
            cout << "Invalid choice" << endl;
      }
   }
   return 0;
}

void insert(int s, int ch) {
   if (t == 1) {  // Inserting the root element
      tree[t++] = ch;
      return;
   }
   int x = search1(s, ch);
   if (tree[x] == -1) {
      tree[x] = ch;
      t++;
   } else if (tree[x] > ch) {
      tree[2 * x] = ch;
   } else {
      tree[2 * x + 1] = ch;
   }
}

void delte(int x) {
   if (tree[2 * x] == -1 && tree[2 * x + 1] == -1) {
      tree[x] = -1;
   } else if (tree[2 * x] == -1) {
      tree[x] = tree[2 * x + 1];
      tree[2 * x + 1] = -1;
   } else if (tree[2 * x + 1] == -1) {
      tree[x] = tree[2 * x];
```

```
        tree[2 * x] = -1;
    } else {
        tree[x] = tree[2 * x];
        delte(2 * x);
    }
    t--;
}

int search(int s) {
    if (t == 1) {
        cout << "No element in tree.";
        return -1;
    }
    if (tree[s] == -1)
        return -1;
    if (tree[s] > x)
        return search(2 * s);
    else if (tree[s] < x)
        return search(2 * s + 1);
    else
        return s;
}

void display(int s) {
    if (t == 1) {
        cout << "No element in tree.";
        return;
    }
    for (int i = 1; i < 40; i++) {
        if (tree[i] == -1)
            cout << " ";
        else
            cout << tree[i] << " ";
    }
```

```
}

int search1(int s, int ch) {
    if (tree[s] == -1)
        return s;
    if (tree[s] > ch)
        return search1(2 * s, ch);
    else
        return search1(2 * s + 1, ch);
}
```

8 Implement Graph ins.,del.,trav.

```
#include <iostream>
#include <list>
#include <map>
#include <queue>
#include <set>

using namespace std;

class Graph {
    map<int, list<int>> adjlist;

public:
    void insertEdge(int u, int v, bool bidirectional = true) {
        adjlist[u].push_back(v);
        if (bidirectional) {
            adjlist[v].push_back(u);
        }
    }

    void deleteEdge(int u, int v, bool bidirectional = true) {
        adjlist[u].remove(v);
        if (bidirectional) {
            adjlist[v].remove(u);
```

```
    }
}

void DFS(int node) {
    map<int, bool> visited;
    DFSHelper(node, visited);
    cout << endl;
}

void DFSHelper(int node, map<int, bool>& visited) {
    visited[node] = true;
    cout << node << " ";
    for (int neighbor : adjlist[node]) {
        if (!visited[neighbor]) {
            DFSHelper(neighbor, visited);
        }
    }
}

void BFS(int start) {
    map<int, bool> visited;
    queue<int> q;
    q.push(start);
    visited[start] = true;

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

        for (int neighbor : adjlist[node]) {
            if (!visited[neighbor]) {
                q.push(neighbor);
                visited[neighbor] = true;
```

```
            }
        }
    }
    cout << endl;
}

void displayGraph() {
    for (auto node : adjlist) {
        cout << node.first << " -> ";
        for (int neighbor : node.second) {
            cout << neighbor << " ";
        }
        cout << endl;
    }
}
};

int main() {
    Graph g;
    g.insertEdge(1, 2);
    g.insertEdge(1, 3);
    g.insertEdge(2, 4);
    g.insertEdge(3, 4);
    g.insertEdge(4, 5);

    cout << "Graph after insertion:" << endl;
    g.displayGraph();

    cout << "DFS Traversal starting from node 1: " << endl;
    g.DFS(1);

    cout << "BFS Traversal starting from node 2: " << endl;
    g.BFS(2);
```

```
    g.deleteEdge(3, 4);
    cout << "Graph after deletion of edge 3-4:" << endl;
    g.displayGraph();

    return 0;
}
```

9 Implement Huffman Coding

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

struct Node {
    char ch;
    int freq;
    Node* left;
    Node* right;

    Node(char ch, int freq) {
        left = right = nullptr;
        this->ch = ch;
        this->freq = freq;
    }
};

// Custom comparator for priority queue
struct compare {
    bool operator()(Node* left, Node* right) {
        return left->freq > right->freq;
    }
};

// Recursive function to print Huffman Codes
```

```
void printCodes(Node* root, string str, unordered_map<char, string>& huffmanCode) {
    if (!root) return;

    if (!root->left && !root->right) {  // Leaf node
        huffmanCode[root->ch] = str;
    }

    printCodes(root->left, str + "0", huffmanCode);
    printCodes(root->right, str + "1", huffmanCode);
}

// Function to build the Huffman Tree and print codes
void buildHuffmanTree(string text) {
    unordered_map<char, int> freq;
    for (char ch : text) {
        freq[ch]++;
    }

    // Create a priority queue to store live nodes of Huffman tree
    priority_queue<Node*, vector<Node*>, compare> pq;

    // Create a leaf node for each character and add it to the priority queue
    for (auto pair : freq) {
        pq.push(new Node(pair.first, pair.second));
    }

    // Iterate until the size of the queue is 1
    while (pq.size() != 1) {
        Node* left = pq.top();
        pq.pop();
        Node* right = pq.top();
        pq.pop();

        // Create a new internal node with a frequency equal to the sum of the two nodes' frequencies
```

```
        int sum = left->freq + right->freq;
        Node* top = new Node('\0', sum);  // '\0' as a placeholder for internal nodes
        top->left = left;
        top->right = right;
        pq.push(top);
    }

    // Root of the Huffman Tree
    Node* root = pq.top();

    // Traverse the Huffman Tree and store Huffman Codes in a map
    unordered_map<char, string> huffmanCode;
    printCodes(root, "", huffmanCode);

    cout << "Huffman Codes are:\n";
    for (auto pair : huffmanCode) {
        cout << pair.first << " : " << pair.second << "\n";
    }

    cout << "\nEncoded string:\n";
    for (char ch : text) {
        cout << huffmanCode[ch];
    }
    cout << "\n";
}

int main() {
    string text;
    cout << "Enter a string to encode: ";
    cin >> text;

    buildHuffmanTree(text);

    return 0;
```

```
}
```

10 Create basic hash table for ins.,del.,trav.

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

class HashTable {
private:
    int* table;
    int size;

public:
    // Constructor to initialize the hash table with a given size
    HashTable(int s) {
        size = s;
        table = new int[size];
        for (int i = 0; i < size; i++) {
            table[i] = -1; // Initialize all entries to -1 (indicating empty slots)
        }
    }

    // Hash function to map keys to table indices
    int hashFunction(int key) {
        return key % size;
    }

    // Insert a key into the hash table
    void insert(int key) {
        int index = hashFunction(key);
        table[index] = key;
    }

    // Delete a key from the hash table
    void deleteKey(int key) {
        int index = hashFunction(key);
```

```
        if (table[index] == key) {
            table[index] = -1; // Mark the slot as empty
        }
    }

    // Traverse and display the hash table
    void traverse() {
        for (int i = 0; i < size; i++) {
            if (table[i] != -1) {
                cout << "Index " << i << ": " << table[i] << endl;
            }
        }
    }

    // Destructor to clean up dynamically allocated memory
    ~HashTable() {
        delete[] table;
    }
};

int main() {
    cout << "Ankit Vishwakarma\n";
    HashTable ht(10);  // Create a hash table with size 10
    ht.insert(23);
    ht.insert(34);
    ht.insert(45);
    ht.traverse();
    ht.deleteKey(34);
    ht.traverse();

    return 0;
}
```