1) Write a C++ code for implementation of stack push and pop operation.

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

int main() {
    stack<int> s;
    s.push(1);
    s.push(2);
    s.push(3);
    cout << "Top element: " << s.top() << endl;
    s.pop();
    cout << "Top after pop: " << s.top() << endl;
    return 0;
}
```

2) Write a C++ code for balanced parentheses.

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

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

```
            if ((ch == ')' && top != '(') ||
                (ch == ']' && top != '[') ||
                (ch == '}' && top != '{'))
                return false;
            s.pop();
        }
    }
    return s.empty();
}

int main() {
    string expr = "{[()]}";
    cout << (isBalanced(expr) ? "Balanced" : "Not Balanced") << endl;
    return 0;
}
```

3) Write a C++ code for implementing queue using two stacks.

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

class Queue {
    stack<int> s1, s2;
public:
    void enqueue(int x) {
        s1.push(x);
    }
    int dequeue() {
        if (s2.empty()) {
```

```
            while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }
        if (s2.empty()) throw runtime_error("Queue is empty");
        int val = s2.top();
        s2.pop();
        return val;
    }
};

int main() {
    Queue q;
    q.enqueue(1);
    q.enqueue(2);
    cout << q.dequeue() << endl;
    q.enqueue(3);
    cout << q.dequeue() << endl;
    return 0;
}
```

4) Write a C++ code for implementing queue front and rear operation.

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

int main() {
```

```
    queue<int> q;
    q.push(10);
    q.push(20);
    cout << "Front: " << q.front() << endl;
    cout << "Rear: " << q.back() << endl;
    q.pop();
    cout << "Front after pop: " << q.front() << endl;
    return 0;
}
```

5) Write a C++ code for counting total number of nodes in linked list.

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

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(NULL) {}
};

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

```
int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    cout << "Total nodes: " << countNodes(head) << endl;
    return 0;
}
```

6) Write a C++ code for finding cycle in the linked list.

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

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(NULL) {}
};

bool hasCycle(Node* head) {
    Node *slow = head, *fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
}
```

```
int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = head->next; // Cycle
    cout << (hasCycle(head) ? "Cycle detected" : "No cycle") << endl;
    return 0;
}
```

7) Write a C++ code for reversing linked list.

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

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(NULL) {}
};

Node* reverseList(Node* head) {
    Node *prev = NULL, *curr = head, *next = NULL;
    while (curr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

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

int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head = reverseList(head);
    printList(head);
    return 0;
}

// More answers (8 to 20) will be added in the next update.
```

8) Write a C++ code for implementing bubble sort.

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

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
```

```
        }
    }
}

int main() {
    int arr[] = {5, 1, 4, 2, 8};
    int n = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, n);
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    return 0;
}
```

9) Write a C++ code for implementing selection sort.

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

void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        int min_idx = i;
        for (int j = i + 1; j < n; ++j)
            if (arr[j] < arr[min_idx])
                min_idx = j;
        swap(arr[i], arr[min_idx]);
    }
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
    selectionSort(arr, n);
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    return 0;
}
```

10) Write a C++ code for implementing insertion sort.

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

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

int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    insertionSort(arr, n);
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
```

```
    return 0;
}
```

11) Write a C++ code for implementing quick sort.

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

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

int main() {
```

```
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

12) Write a C++ code for implementing merge sort.

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

void merge(int arr[], 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] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }
```

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

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

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    mergeSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

13) Write a C++ code for implementing linear search.

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

int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key)
```

```
            return i;
        }
    return -1;
}

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int key = 30;
    int n = sizeof(arr) / sizeof(arr[0]);
    int index = linearSearch(arr, n, key);
    if (index != -1)
        cout << "Element found at index " << index;
    else
        cout << "Element not found";
    return 0;
}
```

14) Write a C++ code for implementing binary search.

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

int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] < key)
            low = mid + 1;
```

```
        else
            high = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int key = 40;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, n, key);
    if (result != -1)
        cout << "Element found at index " << result;
    else
        cout << "Element not found";
    return 0;
}
```

15) Write a C++ code for finding height of the BST.

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

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(NULL), right(NULL) {}
};
```

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

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(20);
    root->left->left = new Node(3);
    root->left->right = new Node(7);
    cout << "Height of BST: " << height(root);
    return 0;
}
```

16) Write a C++ code for counting total number of nodes in BST.

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

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(NULL), right(NULL) {}
};

int countNodes(Node* root) {
```

```
    if (root == NULL)
        return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    cout << "Total nodes in BST: " << countNodes(root);
    return 0;
}
```

17) Write a C++ code for implementing BFS.

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

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(NULL), right(NULL) {}
};

void bfs(Node* root) {
    if (!root) return;
    queue<Node*> q;
    q.push(root);
```

```
    while (!q.empty()) {
        Node* temp = q.front();
        q.pop();
        cout << temp->data << " ";
        if (temp->left) q.push(temp->left);
        if (temp->right) q.push(temp->right);
    }
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    bfs(root);
    return 0;
}
```

18) Write a C++ code for implementing DFS.

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

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(NULL), right(NULL) {}
};
```

```
void dfs(Node* root) {
    if (!root) return;
    cout << root->data << " ";
    dfs(root->left);
    dfs(root->right);
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    dfs(root);
    return 0;
}
```

19) Write a C++ code for implementing graph adjacency list and adjacency matrix.

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

int main() {
    int V = 4;

    // Adjacency List
    vector<vector<int>> adjList(V);
```

```
adjList[0].push_back(1);
adjList[0].push_back(2);
adjList[1].push_back(2);
adjList[2].push_back(0);
adjList[2].push_back(3);
adjList[3].push_back(3);

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

// Adjacency Matrix
int adjMatrix[4][4] = {0};
adjMatrix[0][1] = 1;
adjMatrix[0][2] = 1;
adjMatrix[1][2] = 1;
adjMatrix[2][0] = 1;
adjMatrix[2][3] = 1;
adjMatrix[3][3] = 1;

cout << "\nAdjacency Matrix:\n";
for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
        cout << adjMatrix[i][j] << " ";
    }
```

```
        cout << endl;
    }
    return 0;
}
```

20) Write a C++ code to implement hash table.

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

class HashTable {
    int BUCKET;
    list<int>* table;

public:
    HashTable(int V) {
        BUCKET = V;
        table = new list<int>[BUCKET];
    }

    void insert(int key) {
        int index = key % BUCKET;
        table[index].push_back(key);
    }

    void remove(int key) {
        int index = key % BUCKET;
        table[index].remove(key);
    }
```

```
    void display() {
        for (int i = 0; i < BUCKET; i++) {
            cout << i;
            for (int x : table[i])
                cout << " --> " << x;
            cout << endl;
        }
    }
};

int main() {
    HashTable h(7);
    h.insert(10);
    h.insert(20);
    h.insert(15);
    h.insert(7);
    h.display();
    return 0;
}
```