Major Project

1. Detecting whether a Linked List has any Cycle or not.

```
#include <iostream>
// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

bool hasCycle(ListNode *head) {
    if (!head || !head->next) {
        return false; // If the list is empty or has only one node, there's no cycle.
    }

    ListNode *slow = head;
    ListNode *fast = head->next;

    while (slow != fast) {
        if (!fast || !fast->next) {
            return false; // If fast pointer reaches the end, there's no cycle.
        }
        slow = slow->next;
        fast = fast->next->next;
    }

    return true; // If slow and fast pointers meet, there's a cycle.
}

int main() {
    // Example usage
    ListNode *head = new ListNode(3);
    head->next = new ListNode(2);
```

```
    head->next->next = new ListNode(0);
    head->next->next->next = new ListNode(-4);
    head->next->next->next->next = head->next; // Creating a cycle

    bool cycleExists = hasCycle(head);
    if (cycleExists) {
        std::cout << "The linked list has a cycle.\n";
    } else {
        std::cout << "The linked list does not have a cycle.\n";
    }

    // Freeing memory
    ListNode *current = head;
    while (current) {
        ListNode *temp = current;
        current = current->next;
        delete temp;
    }

    return 0;

}

    /*

    sample input :

    3 -> 2 -> 0 -> -4
     ^         |
     |___________|

     sample output:

     The linked list has a cycle.


    */
```

2. Find the Longest diameter in a Binary Tree.

```
#include <iostream>
#include <queue>


struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Function to insert a new node in the binary tree
void insert(TreeNode* root, int val) {
    std::queue<TreeNode*> q;
    q.push(root);

    // Do level order traversal until we find an empty place
    while (!q.empty()) {
        TreeNode* temp = q.front();
        q.pop();

        if (!temp->left) {
            temp->left = new TreeNode(val);
            break;
        } else
            q.push(temp->left);

        if (!temp->right) {
            temp->right = new TreeNode(val);
            break;
        } else
            q.push(temp->right);
    }
}

// Helper function to take input for the binary tree
TreeNode* takeInput() {
```

```
    std::cout << "Enter the number of nodes in the binary tree: ";
    int n;
    std::cin >> n;

    if (n == 0) return nullptr;

    std::cout << "Enter the values of the nodes: ";
    int rootVal;
    std::cin >> rootVal;
    TreeNode* root = new TreeNode(rootVal);

    for (int i = 1; i < n; ++i) {
        int val;
        std::cin >> val;
        insert(root, val);
    }

    return root;
}

// Helper function to calculate the height of a binary tree
int height(TreeNode* root, int& diameter) {
    if (!root) return 0;
    int leftHeight = height(root->left, diameter);
    int rightHeight = height(root->right, diameter);
    diameter = std::max(diameter, leftHeight + rightHeight);
    return 1 + std::max(leftHeight, rightHeight);
}

// Function to find the largest diameter of a binary tree
int diameterOfBinaryTree(TreeNode* root) {
    int diameter = 0;
    height(root, diameter);
    return diameter;
}

int main() {
    // Taking input for the binary tree
    TreeNode* root = takeInput();
```

```
    // Finding the largest diameter of the binary tree
    std::cout << "Largest diameter of the binary tree: " << diameterOfBinaryTree(root) << std::endl;

    // Free memory
    delete root->left->right;
    delete root->left->left;
    delete root->right;
    delete root->left;
    delete root;
    return 0;
}

/*
sample input:
       1
      / \
     2   3
    / \   \
   4   5   6
            \
             7

 sample output:
 Largest diameter of the binary tree: 5

 This means that the longest path between any two nodes in the binary tree is of length 5.
 In the sample tree, the path with maximum length is from node 4 to node 7.


 */
```