Name: Vinayak Sharma

UID: 22BCS15365

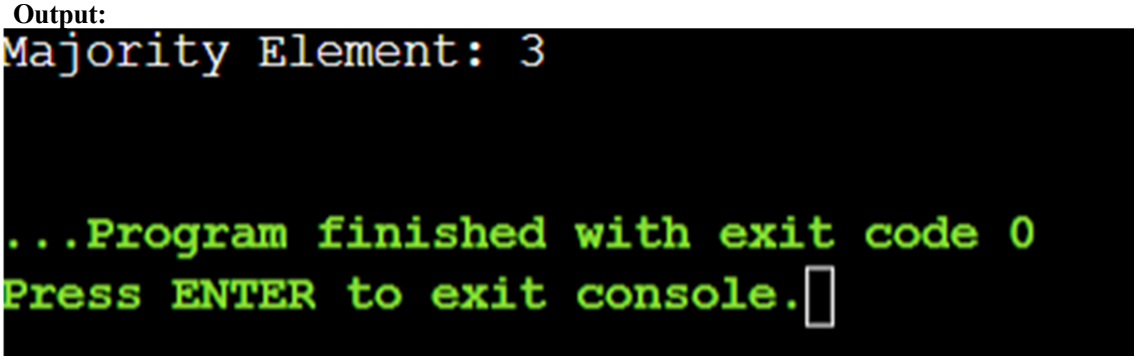
Section: 620-B

Assignment-2

Q 1: Majority Elements

```
#include <iostream>
#include <vector>
using namespace std;
int majorityElement(vector<int>& nums) {
    int count = 0;
    int candidate = 0;
    for (int num : nums) {
        if (count == 0) {
            candidate = num;
        }
        count += (num == candidate) ? 1 : -1;
    }
    return candidate;
}
int main() {
    vector<int> nums = {3, 2, 3};
    cout << "Majority Element: " << majorityElement(nums) << endl;
    return 0;
}
```

Output:

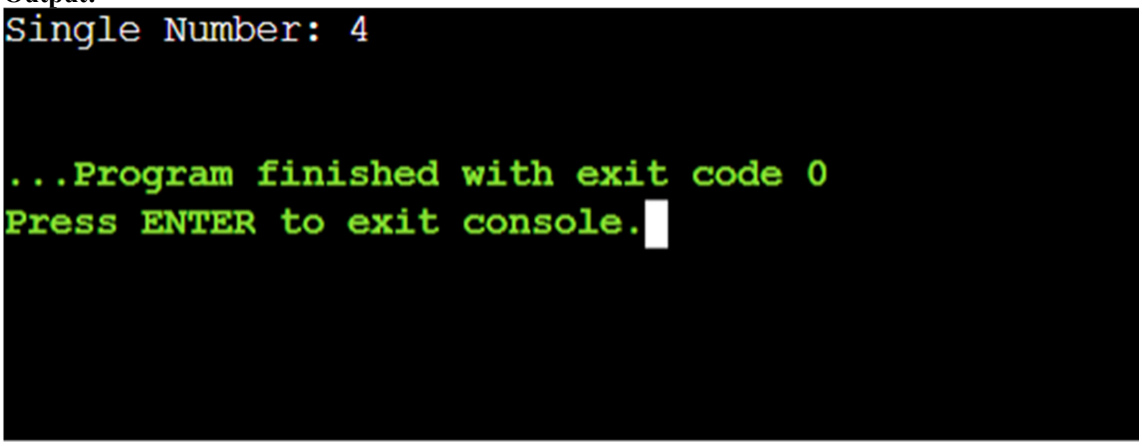


Question 2. Single Number

```
#include <iostream>
```

```
using namespace std;
int singleNumber(int nums[], int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        result ^= nums[i];
    }
    return result;
}
int main() {
    int nums[] = {4, 1, 2, 1, 2};
    int n = sizeof(nums) / sizeof(nums[0]);
    cout << "Single Number: " << singleNumber(nums, n) << endl;
    return 0;
}
```

Output:



Question 3 Convert Sorted Array to Binary Search Tree

```
#include <iostream>
#include <vector>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
TreeNode* sortedArrayToBSTHelper(vector<int>& nums, int left, int right) {
    if (left > right) return NULL;
    int mid = left + (right - left) / 2;
    TreeNode* root = new TreeNode(nums[mid]);
```

```
    root->left = sortedArrayToBSTHelper(nums, left, mid - 1);
    root->right = sortedArrayToBSTHelper(nums, mid + 1, right);
    return root;
}
TreeNode* sortedArrayToBST(vector<int>& nums) {
    return sortedArrayToBSTHelper(nums, 0, nums.size() - 1);
}
void printInOrder(TreeNode* root) {
    if (root == NULL) return;
    printInOrder(root->left);
    cout << root->val << " ";
    printInOrder(root->right);
}
int main() {
    vector<int> nums = {-10, -3, 0, 5, 9};
    TreeNode* root = sortedArrayToBST(nums);
    cout << "In-order traversal of the constructed BST: ";
    printInOrder(root); // Output should be: -10 -3 0 5 9
    cout << endl;
    return 0;
}
```

Output:

```
In-order traversal of the constructed BST: -10 -3 0 5 9


...Program finished with exit code 0
Press ENTER to exit console.
```

Q4. Merge Two Sorted Lists

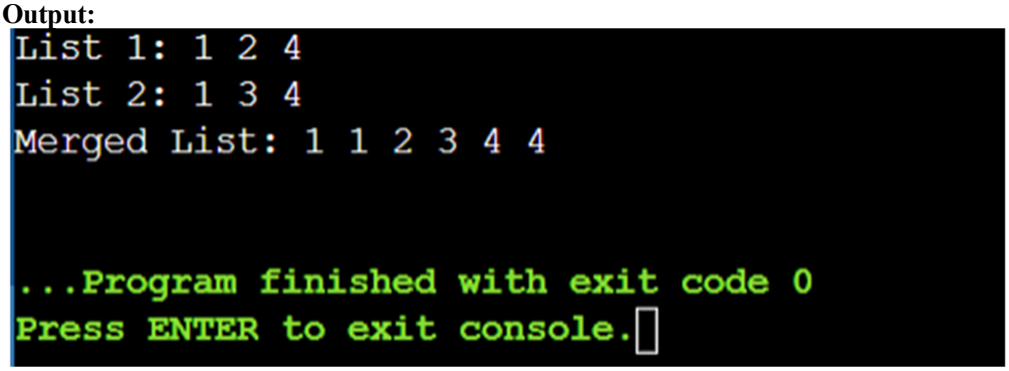
```
#include  <iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(NULL) {}
```

```
};
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    if (!list1) return list2;
    if (!list2) return list1;

    if (list1->val < list2->val) {
        list1->next = mergeTwoLists(list1->next, list2);
        return list1;
    } else {
        list2->next = mergeTwoLists(list1, list2->next);
        return list2;
    }
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}
ListNode* createList(int arr[], int n) {
    if (n == 0) return NULL;
    ListNode* head = new ListNode(arr[0]);
    ListNode* current = head;
    for (int i = 1; i < n; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
    }
    return head;
}
int main() {
    int arr1[] = {1, 2, 4};
    int arr2[] = {1, 3, 4};
    ListNode* list1 = createList(arr1, 3);
    ListNode* list2 = createList(arr2, 3);
    cout << "List 1: ";
    printList(list1);
    cout << "List 2: ";
    printList(list2);
    ListNode* mergedList = mergeTwoLists(list1, list2);
```

```
    cout << "Merged List: ";
    printList(mergedList);
    return 0;
}
```

Output:



Q5. LinkedList Cycle

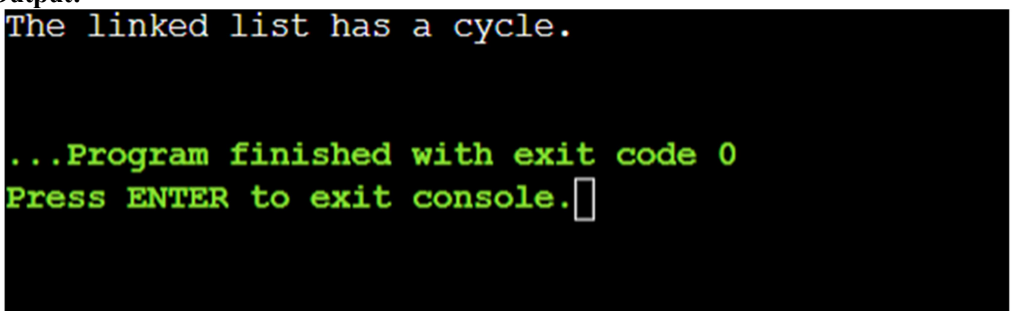
```
#include <iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(NULL) {}
};
bool hasCycle(ListNode* head) {
    if (!head || !head->next) return false;
    ListNode* slow = head;
    ListNode* fast = head;

    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;

        if (slow == fast) return true;
    }
    return false;
}
ListNode* createListWithCycle(int arr[], int n, int pos) {
    if (n == 0) return NULL;
    ListNode* head = new ListNode(arr[0]);
```

```
    ListNode* current = head;
    ListNode* cycleNode = NULL;
    for (int i = 1; i < n; ++i) {
        current->next = new ListNode(arr[i]);
        current = current->next;
        if (i == pos) cycleNode = current;
    }
    if (pos >= 0) current->next = cycleNode;
    return head;
}
int main() {
    int arr[] = {3, 2, 0, -4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int pos = 1;
    ListNode* head = createListWithCycle(arr, n, pos);
    if (hasCycle(head)) {
        cout << "The linked list has a cycle." << endl;
    } else {
        cout << "The linked list does not have a cycle." << endl;
    }
    return 0;
}
```

Output:

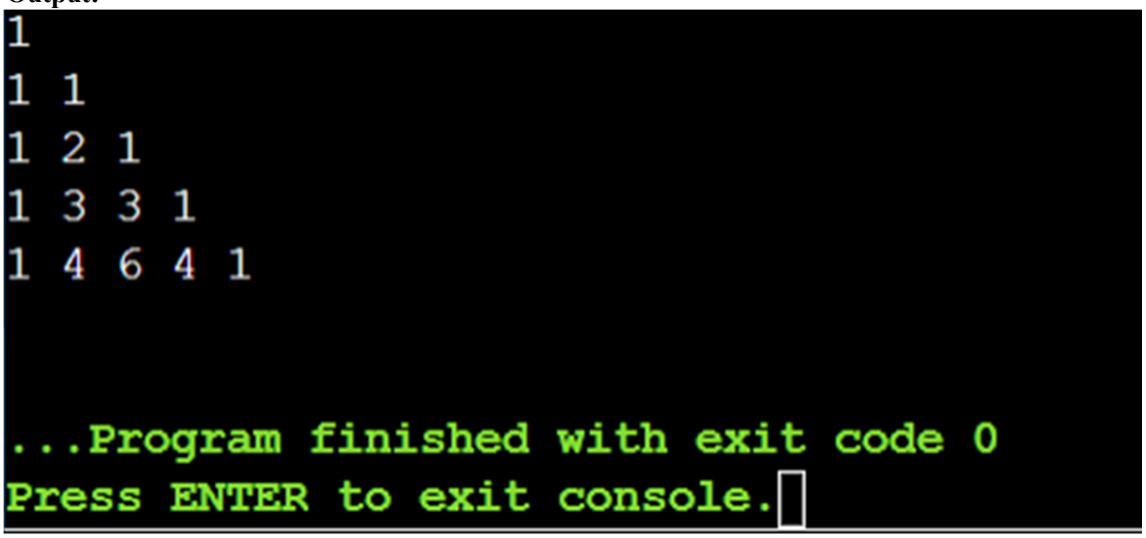


Question 6. Pascal's Triangle

```
#include <iostream>
using namespace std;
void generatePascalTriangle(int numRows) {
    int** triangle = new int*[numRows];
    for (int i = 0; i < numRows; ++i) {
```

```
        triangle[i] = new int[i + 1];
        triangle[i][0] = 1;
        triangle[i][i] = 1;
        for (int j = 1; j < i; ++j) {
            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
        }
    }
    for (int i = 0; i < numRows; ++i) {
        for (int j = 0; j <= i; ++j) {
            cout << triangle[i][j] << " ";
        }
        cout << endl;
    }
    for (int i = 0; i < numRows; ++i) {
        delete[] triangle[i];
    }
    delete[] triangle;
}
int main() {
    int numRows = 5;
    cout << "Pascal's Triangle with " << numRows << " rows:" << endl;
    generatePascalTriangle(numRows);
    return 0;
}
```

Output:

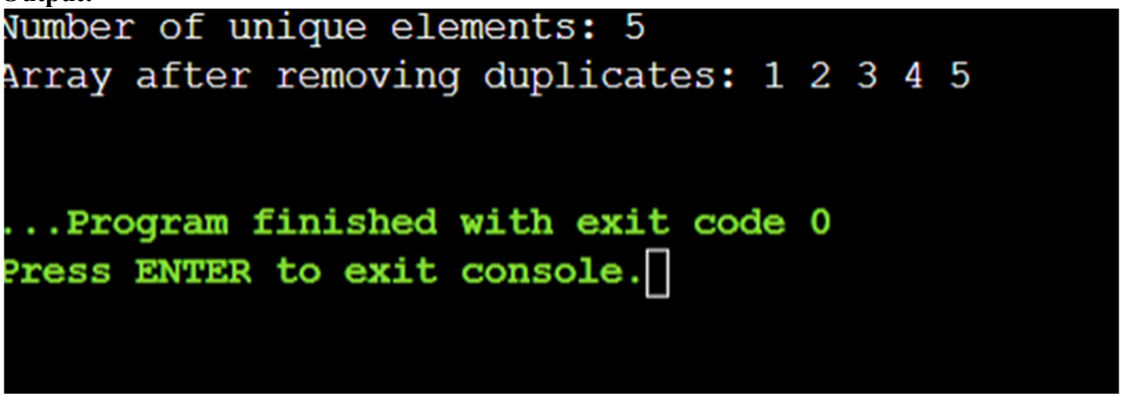


Question 7. Remove Element

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

```
int removeDuplicates(int nums[], int n) {
    if (n == 0) return 0;
    int k = 1;
    for (int i = 1; i < n; ++i) {
        if (nums[i] != nums[k - 1]) {
            nums[k] = nums[i];
            k++;
        }
    }
    return k;
}
int main() {
    int nums[] = {1, 1, 2, 2, 3, 4, 4, 5};
    int n = sizeof(nums) / sizeof(nums[0]);
    int k = removeDuplicates(nums, n);
    cout << "Number of unique elements: " << k << endl;
    cout << "Array after removing duplicates: ";
    for (int i = 0; i < k; ++i) {
        cout << nums[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Output:

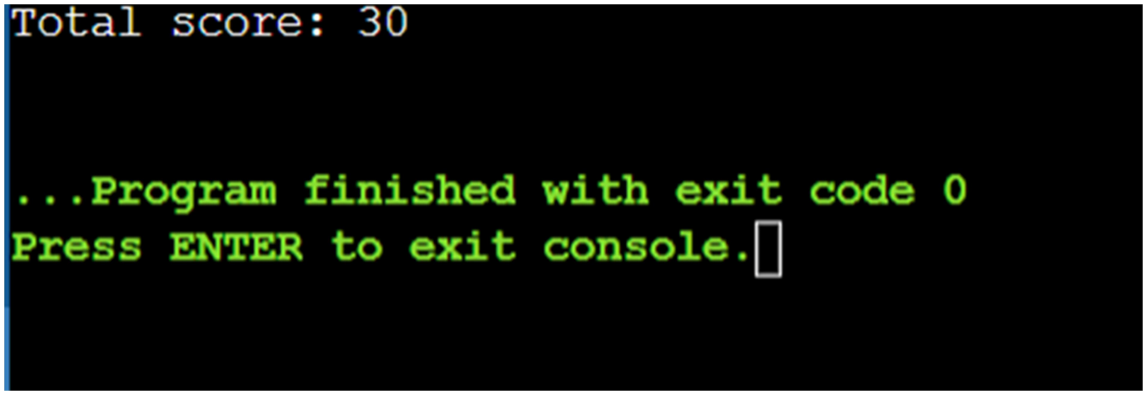


Question 8. Baseball Game

```
#include <iostream>
#include <stack>
#include <string>
```

```
#include <vector>
using namespace std;
int calPoints(vector<string>& operations) {
    stack<int> scores;
    int total = 0;
    for (const string& op : operations) {
        if (op == "+") {
            int top = scores.top();
            scores.pop();
            int newScore = top + scores.top();
            scores.push(top);
            scores.push(newScore);
            total += newScore;
        } else if (op == "D") {
            int newScore = 2 * scores.top();
            scores.push(newScore);
            total += newScore;
        } else if (op == "C") {
            total -= scores.top();
            scores.pop();
        } else {
            int newScore = stoi(op);
            scores.push(newScore);
            total += newScore;
        }
    }
    return total;
}
int main() {
    vector<string> operations = {"5", "2", "C", "D", "+"};
    cout << "Total score: " << calPoints(operations) << endl;
    return 0;
}
```

Output:

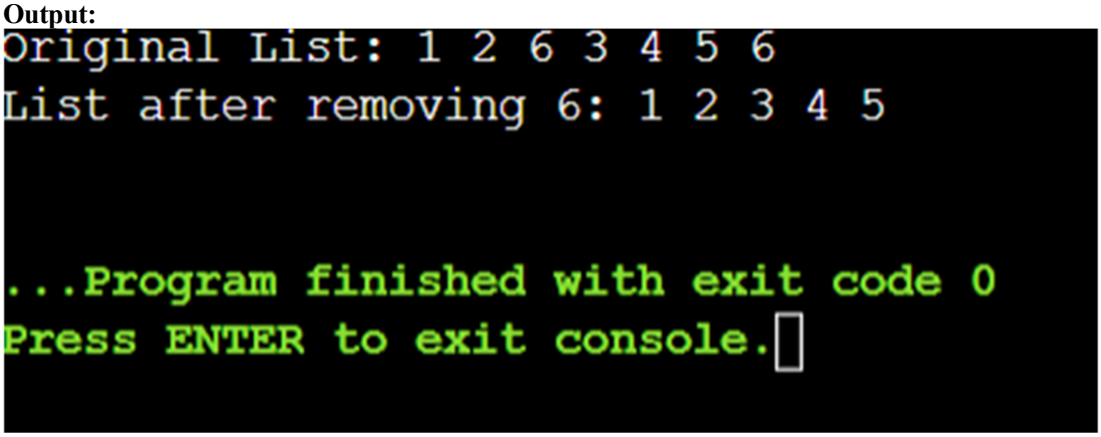


Q9. Remove Linked List Elements

```
#include  <iostream>
using namespace std;
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(NULL) {}
};
ListNode* removeElements(ListNode* head, int val) {
    ListNode* dummy = new ListNode(0);
    dummy->next = head;
    ListNode* current = dummy;
    while (current->next) {
        if (current->next->val == val) {
            ListNode* temp = current->next;
            current->next = current->next->next;
            delete temp;
        } else {
            current = current->next;
        }
    }
    ListNode* newHead = dummy->next;
    delete dummy;
    return newHead;
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
```

```
    }
    cout << endl;
}
ListNode* createList(int arr[], int size) {
    ListNode* head = new ListNode(arr[0]);
    ListNode* temp = head;
    for (int i = 1; i < size; i++) {
        temp->next = new ListNode(arr[i]);
        temp = temp->next;
    }
    return head;
}
int main() {
    int arr[] = {1, 2, 6, 3, 4, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    ListNode* head = createList(arr, size);
    cout << "Original List: ";
    printList(head);
    head = removeElements(head, 6);
    cout << "List after removing 6: ";
    printList(head);
    return 0;
}
```

Output:



Question 10. Container With Most Water

```
#include <iostream>
#include <vector>
using namespace std;
int maxArea(vector<int>& height) {
    int left = 0;
```

```
    int right = height.size() - 1;
    int max_area = 0;
    while (left < right) {
        int width = right - left;
        int current_height = min(height[left], height[right]);
        int current_area = width * current_height;
        max_area = max(max_area, current_area);

        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }
    return max_area;
}
int main() {
    vector<int> height = {1, 1};
    cout << "The maximum amount of water the container can store is: " << maxArea(height) << endl;
    return 0;
}
```

Output: