

Name: Priya Chauhan

UID: 22BCS15310

Section: 22BCS_IOT_620-A

Date: 20-12-24

DOMAIN WINTER WINNING CAMP-Day(2)

1) Majority Element

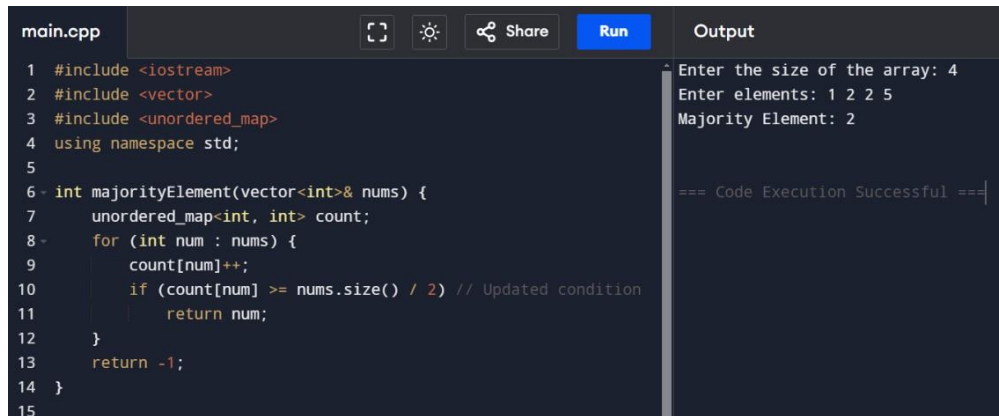
Code:

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

int majorityElement(vector<int>& nums) {
    unordered_map<int, int> count;
    for (int num : nums) {
        count[num]++;
        if (count[num] >= nums.size() / 2) // Updated condition
            return num;
    }
    return -1;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter elements: ";
    for (int& num : nums) cin >> num;
    cout << "Majority Element: " << majorityElement(nums) << endl;
    return 0;
}
```

Output:



The screenshot shows a C++ IDE with a file named 'main.cpp'. The code defines a function 'majorityElement' that uses an unordered_map to count elements in a vector. The main function prompts the user for the size of the array (4) and the elements (1 2 2 5), then calls 'majorityElement' and prints the result (2). The output pane on the right shows the user input and the successful execution message.

```
main.cpp
1 #include <iostream>
2 #include <vector>
3 #include <unordered_map>
4 using namespace std;
5
6 int majorityElement(vector<int>& nums) {
7     unordered_map<int, int> count;
8     for (int num : nums) {
9         count[num]++;
10        if (count[num] >= nums.size() / 2) // Updated condition
11            return num;
12    }
13    return -1;
14 }
15
```

Output

```
Enter the size of the array: 4
Enter elements: 1 2 2 5
Majority Element: 2

=== Code Execution Successful ===
```

2) Single Number

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

```
int singleNumber(vector<int>& nums) {
    int result = 0;
    for (int num : nums) result ^= num;
    return result;
}
```

```
int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter elements: ";
    for (int& num : nums) cin >> num;
    cout << "Single Number: " << singleNumber(nums) << endl;
    return 0;
}
```

Output:

main.cpp	Output
<pre>1 #include <vector> 2 using namespace std; 3 4 int singleNumber(vector<int>& nums) { 5 int result = 0; 6 for (int num : nums) result ^= num; 7 return result; 8 } 9 10 int main() { 11 int n; 12 cout << "Enter the size of the array: "; 13 cin >> n; 14 vector<int> nums(n); 15 cout << "Enter elements: "; 16 for (int& num : nums) cin >> num;</pre>	<pre>Enter the size of the array: 3 Enter elements: 2 1 1 Single Number: 2 === Code Execution Successful ===</pre>

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(nullptr),  
right(nullptr) {}
```

```
};
```

```

TreeNode*
sortedArrayToBST(vector<int>& nums, int left,
int right) {

    if (left > right) return
    nullptr;

    int mid = left + (right
- left) / 2;

    TreeNode* root =
new
TreeNode(nums[mid]);

    root->left =
sortedArrayToBST(num
s, left, mid - 1);

    root->right =
sortedArrayToBST(num
s, mid + 1, right);

    return root;

}

```

```

void
preorder(TreeNode*
root) {

    if (!root) return;

    cout << root->val <<

```

```
";  
  
    preorder(root->left);  
  
    preorder(root->right);  
}
```

```
int main() {  
  
    int n;  
  
    cout << "Enter the  
size of the array: ";  
  
    cin >> n;  
  
    vector<int> nums(n);  
  
    cout << "Enter  
elements in sorted  
order: ";  
  
    for (int& num : nums)  
        cin >> num;  
  
    TreeNode* root =  
sortedArrayToBST(nums, 0, n - 1);  
  
    cout << "Preorder  
traversal of BST: ";  
  
    preorder(root);  
}
```

```

    cout << endl;

    return 0;

}

```

Output:

The screenshot shows a C++ IDE with a file named 'main.cpp'. The code defines a 'TreeNode' struct and a recursive function 'sortedArrayToBST' that converts a sorted array into a Binary Search Tree (BST). The output window shows the program's execution: it prompts for the array size (5), the sorted elements (1 2 3 4 5), and the preorder traversal (3 1 2 4 5). A success message '=== Code Execution Successful ===' is also displayed.

```

main.cpp
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  struct TreeNode {
6      int val;
7      TreeNode* left;
8      TreeNode* right;
9      TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
10 };
11
12 TreeNode* sortedArrayToBST(vector<int>& nums, int left, int
    right) {
13     if (left > right) return nullptr;
14     int mid = left + (right - left) / 2;
15     TreeNode* root = new TreeNode(nums[mid]);
16     root->left = sortedArrayToBST(nums, left, mid - 1);
17     root->right = sortedArrayToBST(nums, mid + 1, right);
18     return root;

```

Output

```

Enter the size of the array: 5
Enter elements in sorted order: 1 2 3 4 5
Preorder traversal of BST: 3 1 2 4 5

=== Code Execution Successful ===

```

4) Merge Two Sorted Lists

```

#include <iostream>
using namespace std;

```

// Definition for singly-linked list

```

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

```

// Function to merge two sorted linked lists

```

ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    if (!l1) return l2;
    if (!l2) return l1;
    if (l1->val < l2->val) {
        l1->next = mergeTwoLists(l1->next, l2);
        return l1;
    } else {
        l2->next = mergeTwoLists(l1, l2->next);

```

```

        return l2;
    }
}

```

// Function to create a linked list from user input

```

ListNode* createList(int n) {
    ListNode* head = nullptr;
    ListNode* tail = nullptr;
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        int val;
        cin >> val;
        ListNode* newNode = new ListNode(val);
        if (!head) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

```

// Function to print a linked list

```

void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

```

```

int main() {
    int n1, n2;
    cout << "Enter size of first sorted list: ";
    cin >> n1;
    ListNode* l1 = createList(n1);

    cout << "Enter size of second sorted list: ";
    cin >> n2;
    ListNode* l2 = createList(n2);

    ListNode* mergedList = mergeTwoLists(l1, l2);
}

```

```

    cout << "Merged Sorted List: ";
    printList(mergedList);

    return 0;
}

```

Output:

main.cpp	Output
<pre> 1 #include <iostream> 2 using namespace std; 3 4 // Definition for singly-linked list 5 struct ListNode { 6 int val; 7 ListNode* next; 8 ListNode(int x) : val(x), next(nullptr) {} 9 }; 10 11 // Function to merge two sorted linked lists 12 ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) { 13 if (!l1) return l2; 14 if (!l2) return l1; </pre>	<pre> Enter size of first sorted list: 3 Enter 3 elements: 1 5 2 Enter size of second sorted list: 2 Enter 2 elements: 9 0 Merged Sorted List: 1 5 2 9 0 === Code Execution Successful === </pre>

5) Reverse Linked List

```

#include <iostream>

using namespace std;

// Definition for singly-linked list
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

// Function to reverse a linked list
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;

```



```
while (head) {  
    ListNode* nextNode = head->next;  
    head->next = prev;  
    prev = head;  
    head = nextNode;  
}  
return prev;  
}
```

// Function to create a linked list from user input

```
ListNode* createList(int n) {  
    ListNode* head = nullptr;  
    ListNode* tail = nullptr;  
    cout << "Enter " << n << " elements: ";  
    for (int i = 0; i < n; i++) {  
        int val;  
        cin >> val;  
        ListNode* newNode = new ListNode(val);  
        if (!head) {  
            head = tail = newNode;  
        } else {  
            tail->next = newNode;  
            tail = newNode;  
        }  
    }  
    return head;  
}
```

// Function to print a linked list

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

```
int main() {  
    int n;  
    cout << "Enter size of the list: ";  
    cin >> n;  
    ListNode* head = createList(n);  
  
    cout << "Original List: ";  
    printList(head);  
  
    head = reverseList(head);  
  
    cout << "Reversed List: ";  
    printList(head);  
  
    return 0;  
}
```

Output:

main.cpp	Output
<pre>1 #include <iostream> 2 using namespace std; 3 4 // Definition for singly-linked list 5 struct ListNode { 6 int val; 7 ListNode* next; 8 ListNode(int x) : val(x), next(nullptr) {} 9 }; 10 11 // Function to reverse a linked list 12 ListNode* reverseList(ListNode* head) { 13 ListNode* prev = nullptr; 14 while (head) { 15 ListNode* nextNode = head->next;</pre>	<pre>Enter size of the list: 5 Enter 5 elements: 1 5 2 9 0 Original List: 1 5 2 9 0 Reversed List: 0 9 2 5 1 === Code Execution Successful ===</pre>

6) Pascals triangle

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

```
// Function to generate Pascal's Triangle
vector<vector<int>> generatePascalsTriangle(int numRows) {
    vector<vector<int>> triangle(numRows);
    for (int i = 0; i < numRows; ++i) {
        triangle[i].resize(i + 1, 1);
        for (int j = 1; j < i; ++j) {
            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
        }
    }
    return triangle;
}
```

```
int main() {
    int numRows;
    cout << "Enter the number of rows for Pascal's Triangle: ";
    cin >> numRows;

    vector<vector<int>> triangle = generatePascalsTriangle(numRows);

    // Calculate the maximum width for spacing
    int maxWidth = triangle[numRows - 1].size() * 4;
```

```

for (int i = 0; i < numRows; ++i) {
    // Print leading spaces for alignment
    int leadingSpaces = (maxWidth - (triangle[i].size() * 4)) / 2;
    cout << string(leadingSpaces, ' ');

    // Print the current row of Pascal's Triangle
    for (int num : triangle[i]) {
        cout << setw(4) << num;
    }
    cout << endl;
}

return 0;
}

```

Output :

The screenshot shows a web browser window with the URL `programiz.com/cpp-programming/online-compiler/`. The page features the Programiz logo and a banner for "Premium Coding Courses by Programiz". Below the banner, there is a code editor with a file named `main.cpp`. The code defines a function `generatePascalsTriangle` that takes the number of rows as input and returns a vector of vectors representing Pascal's Triangle. The `main` function prompts the user to enter the number of rows (5) and calls the `generatePascalsTriangle` function. The output window shows the resulting Pascal's Triangle for 5 rows, with each number right-aligned in a 4-space width. The output is as follows:

```

Enter the number of rows for Pascal's Triangle: 5
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1

```

Below the output, it says "=== Code Execution Successful ===". The browser's taskbar at the bottom shows the time as 8:57 PM on 12/20/2024.

7) Container With Most Water

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

int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int maxArea = 0;
    while (left < right) {
        maxArea = max(maxArea, min(height[left], height[right]) * (right - left));
        if (height[left] < height[right]) ++left;
        else --right;
    }
    return maxArea;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> height(n);
    cout << "Enter elements (heights): ";
    for (int& h : height) cin >> h;
    cout << "Maximum water area: " << maxArea(height) << endl;
    return 0;
}
```

Output :

main.cpp	Output
<pre>1 #include <iostream> 2 #include <vector> 3 using namespace std; 4 5 int maxArea(vector<int>& height) { 6 int left = 0, right = height.size() - 1; 7 int maxArea = 0; 8 while (left < right) { 9 maxArea = max(maxArea, min(height[left], height[right]) 10 * (right - left)); 11 if (height[left] < height[right]) ++left; 12 else --right; 13 } 14 return maxArea; 15 }</pre>	<pre>Enter the size of the array: 5 Enter elements (heights): 1 5 2 9 0 Maximum water area: 10 === Code Execution Successful ===</pre>

8) Remove Duplicates from Array

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

int removeDuplicates(vector<int>& nums) {
    set<int> unique(nums.begin(), nums.end());
    nums.assign(unique.begin(), unique.end());
    return unique.size();
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter elements: ";
    for (int& num : nums) cin >> num;
    int uniqueCount = removeDuplicates(nums);
    cout << "Number of unique elements: " << uniqueCount << endl;
    cout << "Array after removing duplicates: ";
    for (int num : nums) cout << num << " ";
    cout << endl;
    return 0;
}
```

Output :

main.cpp	Run	Output
<pre>1 #include <iostream> 2 #include <vector> 3 #include <set> 4 using namespace std; 5 6 int removeDuplicates(vector<int>& nums) { 7 set<int> unique(nums.begin(), nums.end()); 8 nums.assign(unique.begin(), unique.end()); 9 return unique.size(); 10 } 11 12 int main() { 13 int n; 14 cout << "Enter the size of the array: "; 15 cin >> n;</pre>		<pre>Enter the size of the array: 5 Enter elements: 1 2 2 3 4 Number of unique elements: 4 Array after removing duplicates: 1 2 3 4 === Code Execution Successful ===</pre>

9) Cherry Pickup II (Dynamic Programming Solution)

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

// Function to calculate the maximum cherries that can be picked
int cherryPickup(vector<vector<int>>& grid) {
    int rows = grid.size();
    int cols = grid[0].size();

    // 3D DP table: dp[row][col1][col2] represents the maximum cherries
    // collected
    // by two robots starting from (row, col1) and (row, col2)
    vector<vector<vector<int>>> dp(rows, vector<vector<int>>(cols,
    vector<int>(cols, 0)));

    // Base case: Last row, both robots can collect cherries at their positions
    for (int col1 = 0; col1 < cols; ++col1) {
        for (int col2 = 0; col2 < cols; ++col2) {
            if (col1 == col2) {
                dp[rows - 1][col1][col2] = grid[rows - 1][col1];
            } else {
                dp[rows - 1][col1][col2] = grid[rows - 1][col1] + grid[rows - 1][col2];
            }
        }
    }

    // Fill the DP table from bottom to top
    for (int row = rows - 2; row >= 0; --row) {
        for (int col1 = 0; col1 < cols; ++col1) {
            for (int col2 = 0; col2 < cols; ++col2) {
                int maxCherries = 0;

                // Try all possible moves for both robots
                for (int move1 = -1; move1 <= 1; ++move1) {
                    for (int move2 = -1; move2 <= 1; ++move2) {
                        int newCol1 = col1 + move1;
                        int newCol2 = col2 + move2;

                        if (newCol1 >= 0 && newCol1 < cols && newCol2 >= 0 &&
                        newCol2 < cols) {
```

```

        maxCherries = max(maxCherries, dp[row +
1][newCol1][newCol2]);
    }
}
}

    if (col1 == col2) {
        dp[row][col1][col2] = grid[row][col1] + maxCherries;
    } else {
        dp[row][col1][col2] = grid[row][col1] + grid[row][col2] +
maxCherries;
    }
}
}
}

// Maximum cherries collected starting from the top row
return dp[0][0][cols - 1];
}

```

```

int main() {
    int rows, cols;
    cout << "Enter the number of rows and columns: ";
    cin >> rows >> cols;

    vector<vector<int>> grid(rows, vector<int>(cols));
    cout << "Enter the grid values row by row:\n";
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            cin >> grid[i][j];
        }
    }

    int result = cherryPickup(grid);
    cout << "Maximum cherries collected: " << result << endl;

    return 0;
}

```


Output :

main.cpp	Output
<pre>1 #include <iostream> 2 #include <vector> 3 #include <algorithm> 4 using namespace std; 5 6 // Function to calculate the maximum cherries that can be picked 7 int cherryPickup(vector<vector<int>>& grid) { 8 int rows = grid.size(); 9 int cols = grid[0].size(); 10 11 // 3D DP table: dp[row][col1][col2] represents the maximum cherries collected 12 // by two robots starting from (row, col1) and (row, col2) 13 vector<vector<vector<int>>> dp(rows, vector<vector<int> >>(cols, vector<int>(cols, 0))); 14</pre>	<pre>Enter the number of rows and columns: 4 4 Enter the grid values row by row: 3 1 1 1 2 5 1 2 1 5 5 1 2 1 1 2 Maximum cherries collected: 25 === Code Execution Successful ===</pre>

10) Valid Sudoku

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

```
bool isValidSudoku(vector<vector<char>>& board) {
    for (int i = 0; i < 9; ++i) {
        unordered_set<char> rows, cols, box;
        for (int j = 0; j < 9; ++j) {
            if (board[i][j] != '.' && !rows.insert(board[i][j]).second) return false;
            if (board[j][i] != '.' && !cols.insert(board[j][i]).second) return false;
            int boxRow = 3 * (i / 3) + j / 3;
            int boxCol = 3 * (i % 3) + j % 3;
            if (board[boxRow][boxCol] != '.' &&
!box.insert(board[boxRow][boxCol]).second) return false;
        }
    }
    return true;
}

int main() {
    vector<vector<char>> board(9, vector<char>(9));
    cout << "Enter Sudoku board row by row (use '.' for empty cells):\n";
    for (int i = 0; i < 9; ++i) {
        for (int j = 0; j < 9; ++j) cin >> board[i][j];
    }
    cout << (isValidSudoku(board) ? "Valid Sudoku" : "Invalid Sudoku") << endl;
```

```
    return 0;
}
```

Output :

main.cpp	Output
<pre>1 #include <iostream> 2 #include <vector> 3 #include <unordered_set> 4 using namespace std; 5 6 bool isValidSudoku(vector<vector<char>>& board) { 7 for (int i = 0; i < 9; ++i) { 8 unordered_set<char> rows, cols, box; 9 for (int j = 0; j < 9; ++j) { 10 if (board[i][j] != '.' && !rows.insert(board[i][j]).second) return false; 11 if (board[j][i] != '.' && !cols.insert(board[j][i]).second) return false; 12 int boxRow = 3 * (i / 3) + j / 3; 13 int boxCol = 3 * (i % 3) + j % 3; 14 if (board[boxRow][boxCol] != '.' && !box.insert(board[boxRow][boxCol]).second) return false; 15 } 16 } 17 return true; }</pre>	<pre>Enter Sudoku board row by row (use '.' for empty cells): 5 3 . . 7 6 . . 1 9 5 9 8 6 . 8 . . . 6 . . . 3 4 . . 8 . 3 . . 1 7 . . . 2 . . . 6 . 6 2 8 4 1 9 . . 5 . . . 8 . . 7 9 Valid Sudoku === Code Execution Successful ===</pre>