



Student Name: sweta singh

Branch: BE-CSE

UID -22BCS10664

Section/Group: KPIT-901-A

Q1. Majority Element.

Code: //majority Element

```
#include <stdio.h>
```

```
int majorityElement(int* nums, int numsSize) {
```

```
    int candidate = 0, count = 0;
```

```
    for (int i = 0; i < numsSize; i++) {
```

```
        if (count == 0) {
```

```
            candidate = nums[i];
```

```
        }
```

```
        count += (nums[i] == candidate) ? 1 : -1;
```

```
    }
```

```
    return candidate;
```

```
}
```

```
int main() {
```

```
    int nums[] = {3, 2, 3};
```

```
    int numsSize = sizeof(nums) / sizeof(nums[0]);
```

```
    printf("Majority Element: %d\n", majorityElement(nums, numsSize));
```

```
    return 0;
```

```
}
```

Output:

```
input
Majority Element: 3
....Program finished with exit code 0
Press ENTER to exit console.
```



Q2. Single Number .

Code: #include <stdio.h>

```
int singleNumber(int* nums, int numsSize) {  
    int result = 0;  
    for (int i = 0; i < numsSize; i++) {  
        result ^= nums[i];  
    }  
    return result;  
}  
  
int main() {  
    int nums[] = {4, 1, 2, 1, 2};  
    int numsSize = sizeof(nums) / sizeof(nums[0]);  
    printf("Single Number: %d\n", singleNumber(nums, numsSize));  
    return 0;  
}
```

Output:

A screenshot of a console window with a black background and a blue title bar. The title bar contains the word 'input'. The console shows the output 'Single Number: 4' in white text. Below this, there is a green message: '...Program finished with exit code 0' and 'Press ENTER to exit console.' followed by a white cursor. The console window has a standard toolbar with icons for back, forward, search, and other functions.



Q3. Convert sorted Array to Binary Search Trees .

```
Code: #include <stdio.h>
```

```
#include <stdlib.h>
```

```
struct TreeNode {
```

```
    int val;
```

```
    struct TreeNode* left;
```

```
    struct TreeNode* right;
```

```
};
```

```
struct TreeNode* createNode(int val) {
```

```
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
```

```
    node->val = val;
```

```
    node->left = NULL;
```

```
    node->right = NULL;
```

```
    return node;
```

```
}
```

```
struct TreeNode* sortedArrayToBST(int* nums, int left, int right) {
```

```
    if (left > right) {
```

```
        return NULL;
```

```
    }
```

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

```
    struct TreeNode* root = createNode(nums[mid]);
```

```
    root->left = sortedArrayToBST(nums, left, mid - 1);
```

```
    root->right = sortedArrayToBST(nums, mid + 1, right);
```

```
    return root;
```

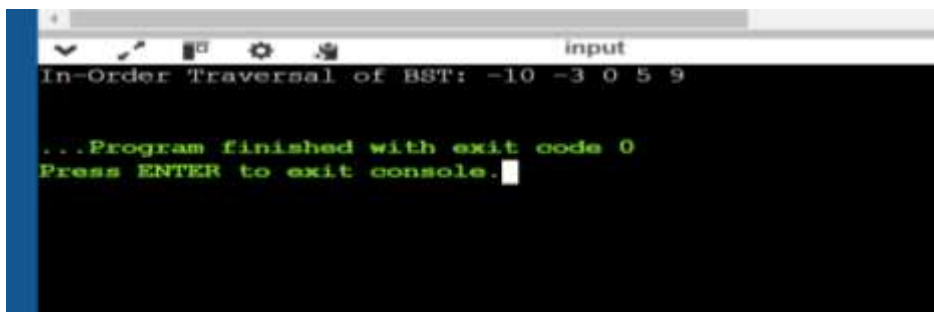
```
}
```

```
struct TreeNode* convertSortedArrayToBST(int* nums, int numsSize) {
```

```
    return sortedArrayToBST(nums, 0, numsSize - 1);
```

```
void printInOrder(struct TreeNode* root) {  
    if (root == NULL) return;  
  
    printInOrder(root->left);  
    printf("%d ", root->val);  
    printInOrder(root->right);  
}  
  
int main() {  
    int nums[] = {-10, -3, 0, 5, 9};  
    int numsSize = sizeof(nums) / sizeof(nums[0]);  
  
    struct TreeNode* root = convertSortedArrayToBST(nums, numsSize);  
  
    printf("In-Order Traversal of BST: ");  
    printInOrder(root);  
    printf("\n");  
  
    return 0;  
}
```

Output :



```
input  
In-Order Traversal of BST: -10 -3 0 5 9  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```



Q4. Merge Two sorted List .

```
Code : #include <stdio.h>
```

```
#include <stdlib.h>
```

```
struct ListNode {
```

```
    int val;
```

```
    struct ListNode* next;
```

```
};
```

```
struct ListNode* createNode(int val) {
```

```
    struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));
```

```
    node->val = val;
```

```
    node->next = NULL;
```

```
    return node;
```

```
}
```

```
struct ListNode* mergeTwoLists(struct ListNode* list1, struct ListNode* list2) {
```

```
    // Base cases
```

```
    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(struct ListNode* head) {
```

```
    struct ListNode* current = head;
```

```
    while (current) {
```



DEPARTMENT OF

Discover. Learn. Empower.

```
        printf("%d ->", current->val);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    // Create first sorted list: 1 -> 2 -> 4
    struct ListNode* list1 = createNode(1);
    list1->next = createNode(2);
    list1->next->next = createNode(4);

    struct ListNode* list2 = createNode(1);
    list2->next = createNode(3);
    list2->next->next = createNode(4);
    struct ListNode* mergedList = mergeTwoLists(list1, list2);
    printf("Merged Sorted List: ");
    printList(mergedList);

    return 0;
}
```

Output :

```
49
Merged Sorted List: 1 -> 1 -> 2 -> 3 -> 4 -> 4 -> NULL

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



Q5. Linked list Cycle .

Code: #include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct ListNode {

int val;

struct ListNode* next;

};

struct ListNode* createNode(int val) {

struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));

node->val = val;

node->next = NULL;

return node;

}

bool hasCycle(struct ListNode* head) {

if (!head || !head->next) return false;

struct ListNode* slow = head;

struct ListNode* fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) return true;

}

return false;



```
// Main function to test the code
int main(){
    struct ListNode* head = createNode(3);
    head->next = createNode(2);
    head->next->next = createNode(0);
    head->next->next->next = createNode(-4);
    head->next->next->next->next = head->next;

    if (hasCycle(head)) {
        printf("The linked list has a cycle.\n");
    } else {
        printf("The linked list does not have a cycle.\n");
    }

    return 0;
}
```

Output:

A screenshot of a terminal window with a dark background. The title bar at the top shows a standard Linux window icon set and the word 'input'. The terminal content shows the output of the program: 'The linked list has a cycle.' followed by a green status message '...Program finished with exit code 0' and a green prompt 'Press ENTER to exit console.' with a white cursor at the end.

```
input
The linked list has a cycle.
...Program finished with exit code 0
Press ENTER to exit console.
```