



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Winter Domain Camp

Student Name: Palak Singla

UID: 22BCS13610

Branch: BE-CSE

Section/Group: 22BCS_IOT_615

Semester: 05

Date of Performance: 20/12/2024

Day 2

Problem 1: Majority Elements

Given an array `nums` of size `n`, return the majority element. The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.

Solution :

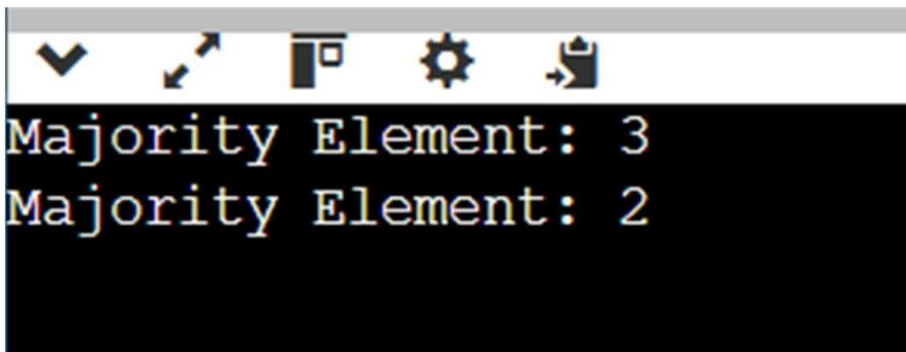
```
#include <iostream>
#include <vector>

int majorityElement(const std::vector<int>&nums) {
    int candidate = 0, count = 0;    // Phase 1: Find the
    candidate      for (int num : nums) {    if (count == 0) {
    candidate = num;    count = 1;    } else if (num ==
    candidate) {    count++;    } else {    count--;
    }
    }

    // Phase 2: Verify the candidate (Optional since problem guarantees majority
    element)    count = 0;    for (int num : nums) {    if (num == candidate) {    count++;
    }
    }
}
```

```
if (count > nums.size() / 2) {  
return candidate;  
}  
  
throw std::runtime_error("No majority element found."); }  
  
int main() {    std::vector<int>nums = {2, 2,  
1, 1, 1, 2, 2};    try {        std::cout<< "Majority Element: "  
<<majorityElement(nums) << std::endl;  
    } catch (const std::exception& e) {  
std::cerr<<e.what() << std::endl;  
    }  
return 0;  
}
```

Output:



```
Majority Element: 3  
Majority Element: 2
```

Problem 2: Pascal's Triangle

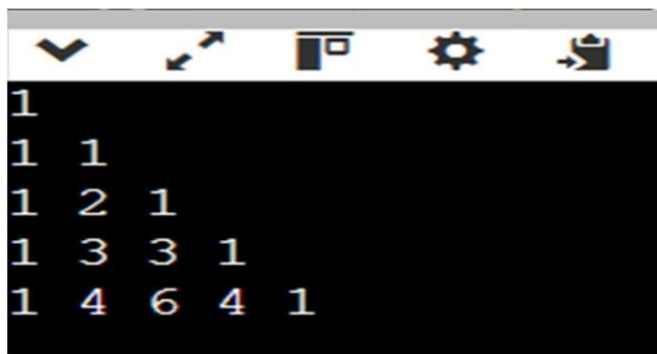
Given an integer numRows, return the first numRows of Pascal's triangle. In Pascal's triangle, each number is the sum of the two numbers directly above it

Solution:

```
#include <iostream>  
#include <vector>
```

```
std::vector<std::vector<int>> generate(int numRows) {
std::vector<std::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 = 5; // Example input  auto result =
generate(numRows);  for (const auto& row :
result) {      for (int num : row)
std::cout<<num<< " ";      std::cout<< "\n";}
return
0;}
```

Output:



```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

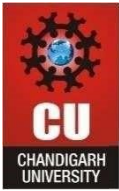
Problem 3: Single Number

Given a non-empty array of integers nums, every element appears twice except for one. Find that single one. You must implement a solution with a linear runtime complexity and use only constant extra space.

Solution:

```
#include <iostream>
#include <vector>

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



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
int main() {    std::vector<int>nums = {4, 1, 2, 1, 2}; // Example input
std::cout<< "Single Number: " <<singleNumber(nums) << std::endl;
return 0; }
```

Output:

A screenshot of a terminal window with a dark background. The text 'Single Number: 4' is displayed in a light blue font. Above the text, there is a toolbar with icons for a checkmark, a cursor, a document, a gear, and a trash can.

Problem 4: Merge Two Sorted Lists

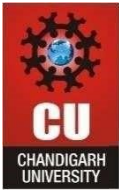
You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.

Solution:

```
#include <iostream>
```

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

```
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;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
    } else {  
        list2->next = mergeTwoLists(list1, list2->next);  
    return list2;  
    }  
}
```

```
void printList(ListNode* head) {  
    while (head) {        std::cout<< head-&br/>>val<< " ";        head = head->next;  
    }  
}
```

```
int main() {  
    ListNode* list1 = new ListNode(1);    list1->next =  
    new ListNode(2);    list1->next->next = new  
    ListNode(4);
```

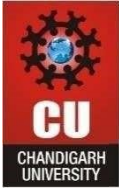
```
    ListNode* list2 = new ListNode(1);    list2->next =  
    new ListNode(3);    list2->next->next = new  
    ListNode(4);
```

```
    ListNode* mergedList = mergeTwoLists(list1, list2);    printList(mergedList);
```

```
    return 0;  
}
```

Output:





Problem 5: Linked List Cycle.

Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter. Return true if there is a cycle in the linked list. Otherwise, return false.

Solution:

```
#include <iostream>

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

bool hasCycle(ListNode* head) {
    if (!head) return false;
    ListNode *slow = head, *fast = head;

    while (fast && fast->next) {        slow = slow->next;
// Move slow pointer by 1 step
fast = fast->next->next;    // Move fast pointer by 2 steps

        if (slow == fast) {            // Cycle
detected            return true;}
    }    return
false; // No
cycle} int
main() {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

// Example 1: Creating a cycle in the list

```
ListNode* head = new ListNode(3);
```

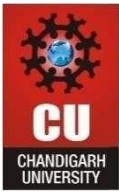
```
head->next = new ListNode(2);
```

```
head->next->next = new ListNode(0);
```

```
head->next->next->next = new ListNode(-4);
```

```
head->next->next->next->next = head->next;
```

// Cycle starts at node with value 2



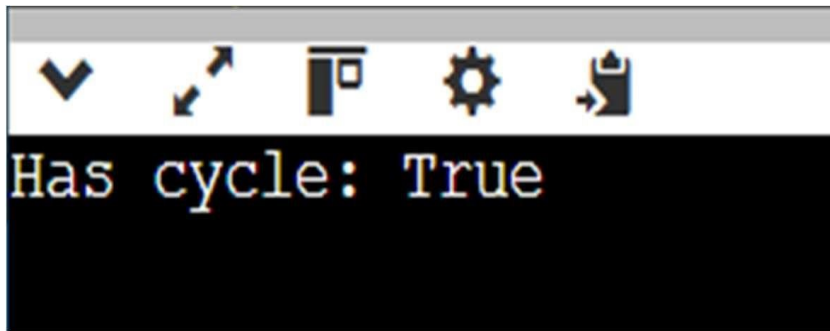
DEPARTMENT OF

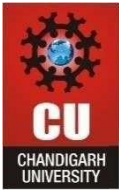
COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
std::cout<< "Has cycle: " << (hasCycle(head) ? "True" : "False") << std::endl;  
  
    return 0;  
}
```

Output:





DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING