DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

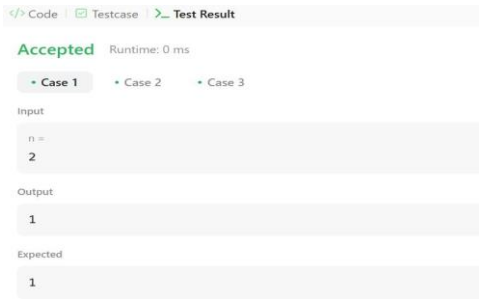
ASSIGNMENT - DAY 3

Name: Tamanna **Section: 22BCS_IOT_615-B**
UID: 22BCS13649 **Branch: BE CSE**
Github Id: Tamanna4141 **Email Id: 22BCS13649@cuchd.in**

VERY EASY

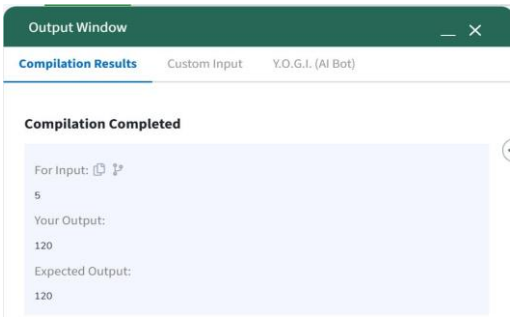
1. Fibonacci series using recursion

```
class Solution {
    public int fib(int n)
        {
        if(n==0||n==1){
            return n;
        }
        return fib(n-1) + fib(n-2);
    }
}
```



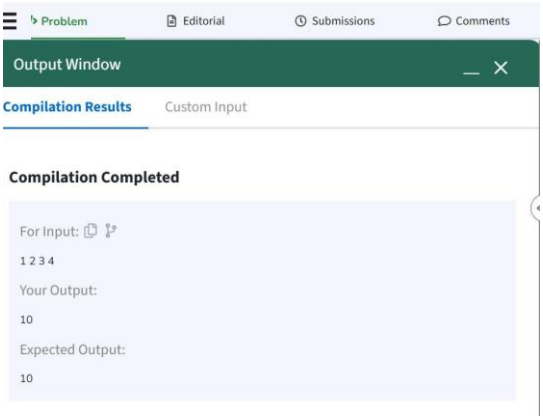
2. Factorial of number using recursion

```
class Solution {
  static int factorial(int
    n) { int result =1;
    for(int i=1;i<=n;i++){
        result = n*factorial(n-1);
    }
    return result;

    }

  }
```



3. Sum of array elements using recursion

```
class Solution {
  int sum(int arr[])
     { int sum = 0;
     for(int
       i=0;i<arr.length;i++)
       { sum = sum + arr[i];
     }
     return sum;
  }
}
```



4. To find reverse of string using recursion

```
import java.io.*;
import java.util.Scanner;

class Solution {
  public static void main(String[] args) {

    String s = "Geeks";
    String r = "";
    char ch;

    for (int i = 0; i < s.length();


      i++) { ch = s.charAt(i);


      r = ch + r;
    }

    System.out.println(r);
  }
}
```

```
Output:   skeeG
```

5. Sum of natural number using recursion

```
import
java.util.*;
import
java.lang.*;

class Solution
{
        public static int recurSum(int n)
        {
                if (n <= 1)
                        return n;
                return n + recurSum(n - 1);
        }


        public static void main(String args[])
        {
                int n = 5;
                System.out.println(recurSum(n));
        }
}
```

```
Output : 15
```

Easy

1. Merge Two Sorted Lists

```
#include <iostream>

using namespace std;

struct ListNode {

   int val;

   ListNode* next;

   ListNode() : val(0), next(nullptr) {}

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

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

};
```

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

    ListNode* dummy = new ListNode(-1);
    ListNode* current = dummy;

    while (list1 && list2) {
        if (list1->val <= list2->val) {
            current->next = list1;
            list1 = list1->next;
        } else {
            current->next = list2;
            list2 = list2->next;
        }
        current = current->next;
    }

    if (list1) current->next = list1;
    if (list2) current->next = list2;

    return dummy->next;
}

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

```
 }
    cout << endl;
}

ListNode* createList(int arr[], int n) {
    if (n == 0) return nullptr;
    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);

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

    return 0;
}
```

OUTPUT:

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

2. Remove Linked List Elements

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

ListNode* removeElements(ListNode* head, int val) {
    ListNode* dummy = new ListNode(-1, 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;
        }
    }

    return dummy->next;
}

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

ListNode* createList(int arr[], int n) {
    if (n == 0) return nullptr;
    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 arr[] = {1, 2, 6, 3, 4, 5, 6};
    int val = 6;

    ListNode* head = createList(arr, 7);

    ListNode* newHead = removeElements(head, val);
    printList(newHead);

    return 0;
}
```

OUTPUT:

```
1 2 3 4 5
```

3. Reverse Linked List

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* current = head;

    while (current) {
        ListNode* nextNode = current->next; // Store next node
        current->next = prev;
        prev = current;
        current = nextNode;
    }

    return prev;
}

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

```
 cout << endl;
}

ListNode* createList(int arr[], int n) {
    if (n == 0) return nullptr;
    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 arr[] = {1, 2, 3, 4, 5};
    ListNode* head = createList(arr, 5);

    ListNode* reversedHead = reverseList(head);
    printList(reversedHead);

    return 0;
}
```

OUTPUT:

```
5 4 3 2 1
```

Medium

1 . Add Two Numbers

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode* next) : val(x), next(next) {}
};

ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode* dummy = new ListNode(-1);
    ListNode* current = dummy;
    int carry = 0;

    while (l1 || l2 || carry) {
        int sum = carry;
```

```
 if (l1) {
        sum += l1->val;
        l1 = l1->next;
    }

    if (l2) {
        sum += l2->val;
        l2 = l2->next;
    }

    carry = sum / 10;
    current->next = new ListNode(sum % 10);
    current = current->next;
  }

  return dummy->next;
}

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

ListNode* createList(int arr[], int n) {
  if (n == 0) return nullptr;
  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[] = {2, 4, 3};
  int arr2[] = {5, 6, 4};

  ListNode* l1 = createList(arr1, 3);
  ListNode* l2 = createList(arr2, 3);

  ListNode* result = addTwoNumbers(l1, l2);
  printList(result);

  return 0;
}
```

OUTPUT:

```
7 0 8
```

2 . Elimination Game

```
#include <iostream>

using namespace std;


int lastRemaining(int n) {

    int left = 1, step = 1;

    bool leftToRight = true;

    int remaining = n;


    while (remaining > 1) {

        if (leftToRight || remaining % 2 == 1) {

            left += step;

        }

        step *= 2;

        remaining /= 2;

        leftToRight = !leftToRight;

    }


    return left;

}

int main() {

    int n = 9; // Example input

    cout << "Last remaining number: " << lastRemaining(n) << endl;

    return 0;

}
```

OUTPUT:

```
Last remaining number: 6
```

Q3 . Predict The Winner

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

bool PredictTheWinner(vector<int>& nums) {
    int n = nums.size();
    vector<vector<int>> dp(n, vector<int>(n, 0));

    for (int i = 0; i < n; i++) {
        dp[i][i] = nums[i];
    }

    for (int len = 2; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            dp[i][j] = max(nums[i] - dp[i + 1][j], nums[j] - dp[i][j - 1]);
        }
    }

    return dp[0][n - 1] >= 0;
}

int main() {
    vector<int> nums = {1, 5, 2}; // Example input
    if (PredictTheWinner(nums)) {
        cout << "Player 1 can win." << endl;
    } else {
        cout << "Player 1 cannot win." << endl;
    }
    return 0;
}
```

OUTPUT:

```
Player 1 cannot win.
```

Hard

Q1 ..Regular Expression Matching

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

```
bool isMatch(string s, string p) {
    int m = s.size(), n = p.size();
    vector<vector<bool>> dp(m + 1, vector<bool>(n + 1, false));

    dp[0][0] = true; // Both string and pattern are empty.

    for (int j = 1; j <= n; j++) {
        if (p[j - 1] == '*') {
            dp[0][j] = dp[0][j - 2]; // '*' matches zero occurrences of the preceding character.
        }
    }

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (p[j - 1] == s[i - 1] || p[j - 1] == '.') {
                dp[i][j] = dp[i - 1][j - 1];
            } else if (p[j - 1] == '*') {
                dp[i][j] = dp[i][j - 2] || (dp[i - 1][j] && (s[i - 1] == p[j - 2] || p[j - 2] == '.'));
            }
        }
    }

    return dp[m][n];
}

int main() {
    string s = "aa";
    string p = "a";

    if (isMatch(s, p)) {
        cout << "True" << endl;
    } else {
        cout << "False" << endl;
    }

    return 0;
}
```

OUTPUT:

```
False
```

Q2. Reverse Nodes in k-Group

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

struct ListNode {
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
```

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

ListNode* reverseKGroup(ListNode* head, int k) {
    if (!head || k == 1) return head; // If no list or k == 1, return the list as is.

    ListNode* dummy = new ListNode(0);
    dummy->next = head;
    ListNode* prevGroupEnd = dummy;
    ListNode* current = head;

    while (current) {
        ListNode* groupStart = current;
        int count = 0;

        // Check if there are k nodes left to reverse.
        while (current && count < k) {
            current = current->next;
            count++;
        }

        if (count == k) {
            ListNode* groupEnd = groupStart;
            ListNode* nextGroupStart = current;
            ListNode* prev = nullptr;

            // Reverse the k nodes.
            while (groupStart != current) {
                ListNode* temp = groupStart->next;
                groupStart->next = prev;
                prev = groupStart;
                groupStart = temp;
            }

            // Connect reversed group with previous and next parts.
            prevGroupEnd->next = prev;
            groupEnd->next = nextGroupStart;
            prevGroupEnd = groupEnd;
        }
    }

    return dummy->next;
}

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

```
ListNode* createList(int arr[], int n) {

if (n == 0) return nullptr;
    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 arr[] = {1, 2, 3, 4, 5};
    int k = 2;
    ListNode* head = createList(arr, 5);

    ListNode* result = reverseKGroup(head, k);
    printList(result);

    return 0;
}
```

OUTPUT:

```
2 1 4 3 5
```

Q3. Wildcard Matching

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

bool isMatch(string s, string p) {
    int m = s.size(), n = p.size();
    vector<vector<bool>> dp(m + 1, vector<bool>(n + 1, false));

    dp[0][0] = true;

    for (int j = 1; j <= n; j++) {
        if (p[j - 1] == '*') {
            dp[0][j] = dp[0][j - 1];
        }
    }

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (p[j - 1] == s[i - 1] || p[j - 1] == '?') {
                dp[i][j] = dp[i - 1][j - 1];
            } else if (p[j - 1] == '*') {
                dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
            }
        }
```

```
    }
 return dp[m][n];
}

int main() {
    string s = "aa";
    string p = "a";

    if (isMatch(s, p)) {
        cout << "True" << endl;
    } else {
        cout << "False" << endl;
    }

    return 0;
}
```

OUTPUT:

```
True
```

Very Hard

Q1. Find the K-th Character in String Game II

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

char findKthCharacter(int k, vector<int>& operations) {
    string word = "a";

    for (int op : operations) {
        int n = word.size();

        if (op == 0) {
            word += word;
        } else if (op == 1) {
            for (int i = 0; i < n; i++) {
                word[i] = (word[i] == 'z') ? 'a' : word[i] + 1;
            }
            word += word;
        }

        if (k <= word.size()) {
            return word[k - 1];
        }
    }

    return word[k - 1];
}
```

```
int main() {
    int k = 5;

vector<int> operations = {0, 0, 0};

    cout << findKthCharacter(k, operations) << endl;

    return 0;
}
```

OUTPUT:

```
a
```

Q2. Maximize Number of Nice Divisors

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

const int MOD = 1e9 + 7;

long long powMod(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int maxNiceDivisors(int primeFactors) {
    if (primeFactors <= 3) return primeFactors;

    long long quotient = primeFactors / 3;
    long long remainder = primeFactors % 3;
    long long result = powMod(3, quotient, MOD);

    if (remainder == 1) {
        result = (result * 4) % MOD;  // 1 left over, we combine it with 3 to form 4.
    } else if (remainder == 2) {
        result = (result * 2) % MOD;  // 2 left over, we multiply by 2.
    }

    return result;
}

int main() {
```

```
int primeFactors = 5;
    cout << maxNiceDivisors(primeFactors) << endl;
    return 0;
}
```

OUTPUT:

```
6
```

Q3. Parsing a Boolean Expression

```
 #include <iostream>

#include <stack>

#include <string>

using namespace std;


bool parseBoolExpr(string expression) {

    stack<bool> s;


    for (int i = expression.size() - 1; i >= 0; i--) {

        char c = expression[i];


        if (c == 't') {

            s.push(true);

        } else if (c == 'f') {

            s.push(false);

        } else if (c == ')') {

            continue;

        } else if (c == '!') {

            bool val = s.top(); s.pop();

            s.push(!val);

        } else if (c == '&') {

            bool result = true;
```

```
while (expression[i] != '(') {
            i--;
        }
        while (expression[i] != ')') {
            if (expression[i] == 't') {
                result = result && true;
            } else if (expression[i] == 'f') {
                result = result && false;
            }
            i--;
        }
        s.push(result);
    } else if (c == '|') {
        bool result = false;
        while (expression[i] != '(') {
            i--;
        }
        while (expression[i] != ')') {
            if (expression[i] == 't') {
                result = result || true;
            } else if (expression[i] == 'f') {
                result = result || false;
            }
            i--;
        }
        s.push(result);
    }
}
```

```
    return s.top();
}
int main() {
    string expression = "&(|(f))";
    cout << parseBoolExpr(expression) << endl;
    return 0;
}
```

OUTPUT:

```
false
```