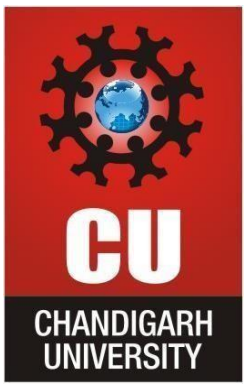

Winter Winning Camp

ASSIGNMENT – 6

Student Name: Ishan Jain

UID: 22BCS13653

Date of Submission: 23/12/2024

Branch: BE-CSE

Section/Group: 615-B

DAY – 6

```
#include <iostream>
```

```
#include <vector>
#include <string>
#include <unordered_map>
#include <stack>
#include <algorithm>
#include <cmath>
#include <queue>
#include <limits.h> using
namespace std;
```

// Q1: Fibonacci Series Using Recursion

```
int fibonacci(int n) {     if (n <= 1) return n;
return fibonacci(n - 1) + fibonacci(n - 2);
}
```

// Q2: Factorial Of Number Using Recursion

```
int factorial(int n) {     if (n ==
0 || n == 1) return 1;     return
n * factorial(n - 1);
}
```

// Q3: Sum of Natural Number Using Recursion

```
int recur_sum(int n) {     if (n
== 0) return 0;     return n +
recur_sum(n - 1);
}
```

// Q4: Sum of Array Elements Using Recursion

```
int sumArray(int arr[], int n) {     if (n <=
0) return 0;     return arr[n - 1] +
sumArray(arr, n - 1);
}
```

// Q5: To Find Reverse Of String Using Recursion

```
string reverseString(string str) {     if (str.empty()) return "";
return str.back() + reverseString(str.substr(0, str.size() - 1));
}
```

// Easy Questions

// Q1: Merge Two Sorted Lists

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

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;
    }
```

```
}
```

// Q2: Remove Linked List Elements

```
ListNode* removeElements(ListNode* head, int val) {
ListNode dummy(0);     dummy.next = head;
ListNode* current = &dummy;     while (current-
>next) {         if (current->next->val == val) {
current->next = current->next->next;
       } else {             current =
current->next;
       }
   }
   return dummy.next;
}
```

// Q3: Reverse Linked List

```
ListNode* reverseList(ListNode* head) {
   ListNode* prev = NULL;
ListNode* current = head;     while
(current) {
       ListNode* nextTemp = current->next;
current->next = prev;         prev = current;
current = nextTemp;
   }     return
prev;
}
```

// Q4: Power Of Three

```
bool isPowerOfThree(int n) {
if (n <= 0) return false;     while
(n % 3 == 0) n /= 3;     return n
== 1;
}
```

// Q5: Palindrome Linked List

```
bool isPalindrome(ListNode* head) {
   if (!head) return true;     ListNode
*slow = head, *fast = head;     while
(fast && fast->next) {         slow =
slow->next;         fast = fast->next-
>next;
   }
   ListNode* prev = NULL;
while (slow) {
       ListNode* nextTemp = slow->next;
slow->next = prev;         prev = slow;
slow = nextTemp;
   }     while (prev) {         if (head->val !=
prev->val) return false;         head = head-
>next;         prev = prev->next;
   }     return
true;
}
```

// Q6: Find the K-th Character in String Game

```
char findKthCharacter(int k) {     string
word = "a";     while (word.length() < k) {
string newWord = "";         for (char c :
word) {             newWord += (c == 'z') ?
'a' : c + 1;
        }
        word += newWord;
    }     return word[k
- 1];
}
```

// Medium Questions

// Q1: Add Two Numbers

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    ListNode dummy(0);
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;
}
```

// Q2: Elimination Game

```
int lastRemaining(int n) {      bool
leftToRight = true;      int remaining = n, step
= 1, head = 1;      while (remaining > 1) {
if (leftToRight || remaining % 2 == 1) {
head += step;
        }
        remaining /= 2;          step
*= 2;          leftToRight =
!leftToRight;
    }      return
head;
}
```

// Q3: Predict The Winner 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;
}
```

// Q4: Find The Winner Of Circular Game

```
int findTheWinner(int n, int k) {     if (n == 1)
return 1;     return (findTheWinner(n - 1, k) + k - 1)
% n + 1;
}
```

// Q5: Minimum non-zero product of an Array Elements

```
int minNonZeroProduct(int p) {     const int MOD = 1e9 + 7;
long long maxNum = (1LL << p) - 1;     long long result =
pow(maxNum, (maxNum - 1) / 2, MOD);     return (result *
(maxNum % MOD)) % MOD;
}
```

// Hard Questions

// Q1: Regular Expression Matching

```
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 - 2];
        }
    }     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];
}
```

// Q2: Reverse Nodes in k-Group

```
ListNode* reverseKGroup(ListNode* head, int k) {
ListNode dummy(0);     dummy.next = head;
    ListNode* prevGroupEnd = &dummy;
    while (true) {
        ListNode* kGroupStart = prevGroupEnd->next;
ListNode* kGroupEnd = prevGroupEnd;
        for (int i = 0; i < k; i++) {
kGroupEnd = kGroupEnd->next;             if
(!kGroupEnd) return dummy.next;
        }
        ListNode* nextGroupStart = kGroupEnd->next;
        kGroupEnd->next = nullptr;         prevGroupEnd-
>next = reverseList(kGroupStart);         kGroupStart-
```

```
>next = nextGroupStart;         prevGroupEnd =
kGroupStart;
    }
}
```

// Q3: Wildcard Matching

```
bool isMatchWildcard(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][j - 1] || dp[i - 1][j];
            }
        }
    }
    return dp[m][n];
}
```

// Q4: Permutation Sequence

```
string getPermutation(int n, int k) {
vector<int> nums;     for (int i = 1;
i <= n; i++) {
nums.push_back(i);
    }
```

```
    k--; // Convert to 0-based index
string result;     int factorial = 1;     for
(int i = 1; i < n; i++) {         factorial *=
i;     }     for (int i = n; i > 0; i--) {
int index = k / factorial;         result +=
to_string(nums[index]);
nums.erase(nums.begin() + index);
k %= factorial;         if (i > 1) {
factorial /= (i - 1);
        }     }
return result;
}
```

// Q5: Basic Calculator

```
int calculate(string s) {
stack<int> nums;
stack<char> ops;     int
num = 0;     char
lastOp = '+';     for
(char c : s) {
        if (isdigit(c)) {             num =
num * 10 + (c - '0');
        } else if (c == '+' || c == '-' || c == '(' || c == ')') {
if (lastOp == '+') nums.push(num);             else if
(lastOp == '-') nums.push(-num);             num = 0;
lastOp = c;             if (c == ')') {                 int sum =
0;                 while (!ops.empty() && ops.top() !=
```

```
'(') {                    sum += nums.top();
nums.pop();
                }
                if (!ops.empty()) ops.pop(); // Remove '('
nums.push(sum);             } else if (c == '(') {
ops.push(c);
            }
        }
    }
    if (lastOp == '+') nums.push(num);
else if (lastOp == '-') nums.push(-num);
int result = 0;     while (!nums.empty()) {
result += nums.top();         nums.pop();
    }     return
result;
}
```