WEEK 4 : STACK, QUEUE & LINKED LIST

➔" Problem 1: The Haunted Castle's Undo Spell "

CODE :-

```
#include <iostream>
#include <stack>
#include <unordered_map>
#include <algorithm>
using namespace std;

pair<int, int> finalPosition() {
   stack<pair<int, int>> s;

   unordered_map<string, pair<int, int>> moveMap = {
      {"NORTH", {0, 1}},
      {"SOUTH", {0, -1}},
      {"EAST", {1, 0}},
      {"WEST", {-1, 0}}
   };
   string move;
   while (cin >> move) {
      // Convert input to uppercase
      transform(move.begin(), move.end(), move.begin(), ::toupper);
      if (move == "Undo") {
         if (!s.empty()) s.pop(); // Undo last move
      } else if (moveMap.find(move) != moveMap.end()) {
         s.push(moveMap[move]); // Push valid move to stack
      } else {
         cout << "Invalid move. Please enter NORTH, SOUTH, EAST, WEST, or UNDO.\n";
      }
   }
   // Compute final position
   int x = 0, y = 0;
   while (!s.empty()) {
      x += s.top().first;
      y += s.top().second;
      s.pop();
   }
   return {x, y};
}

int main() {
   pair<int, int> result = finalPosition();
   cout << "(" << result.first << " , " << result.second << ")\n";
   return 0;
}
```

➔ " Problem 2: The Magic Mirror "

CODE :-

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

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

// Function to reverse k nodes in the linked list
ListNode* reverseKGroup(ListNode* head, int k) {
   ListNode* curr = head;
   int count = 0;
   // Check if there are at least K nodes
   while (curr && count < k) {
      curr = curr->next;
      count++;
   }

   if (count < k) return head; // Not enough nodes, return as is
   // Reverse K nodes
   curr = head;
   ListNode* prev = nullptr;
   ListNode* next = nullptr;
   count = 0;

   while (curr && count < k) {
      next = curr->next;
      curr->next = prev;
      prev = curr;
      curr = next;
      count++;
   }
   // Recursively reverse the next segment
   head->next = reverseKGroup(curr, k);
   return prev; // New head after reversal
}
// Function to print the linked list
void printList(ListNode* head) {
   while (head) {
      cout << head->val << " ";
      head = head->next;
   }
   cout << endl;
}
```

```
// Function to delete the linked list and free memory
void deleteList(ListNode* head) {
   while (head) {
      ListNode* temp = head;
      head = head->next;
      delete temp;
   }
}

int main() {
   int N, K;
   cin >> N >> K;
   ListNode* head = nullptr;
   ListNode* tail = nullptr;

   for (int i = 0; i < N; i++) {
      int value;
      cin >> value;
      ListNode* newNode = new ListNode(value);
      if (!head) head = newNode;
      else tail->next = newNode;
      tail = newNode;
   }
   head = reverseKGroup(head, K);
   printList(head);

   // Free memory
   deleteList(head);
   return 0;
}
```

➔ " Problem 3: The Quantum Printer "

CODE :-

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

vector<int> processJobs(int n, vector<int>& jobs) {
   queue<int> jobQueue; // FIFO queue for jobs
   vector<int> result; // Stores the execution order

   for (int i = 0; i < n; i++) {
      if ((i + 1) % 3 == 0) { // Every 3rd job is processed immediately
         result.push_back(jobs[i]);
      }
```

```
        else {
            jobQueue.push(jobs[i]);   // Other jobs are enqueued in FIFO order
        }
    }

    // Process remaining jobs in FIFO order
    while (!jobQueue.empty()) {
        result.push_back(jobQueue.front());
        jobQueue.pop();
    }
    return result;
}

// Main function to handle input and output
int main() {
    int n;
    cin >> n;     // Read number of jobs

    vector<int> jobs(n);
    for (int i = 0; i < n; i++) {
        cin >> jobs[i];
    }
    vector<int> executionOrder = processJobs(n, jobs);      // Get job execution order

    for (int job : executionOrder) {
        cout << job << " ";
    }
    cout << endl;
    return 0;
}
```

➔ " Problem 4: The Lost Scrolls of Arithmos "

CODE :-

```
#include <bits/stdc++.h>
using namespace std;

// Function to check if a number is prime
bool isPrime(int n) {
    if (n < 2) return false;
    if (n == 2 || n == 3) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0)
            return false;
    }
```

```
    return true;
}

int main() {
    int N, K;
    cin >> N;
    vector<int> stack(N);

    for (int i = 0; i < N; i++) cin >> stack[i];
    cin >> K;
    vector<int> result;
    deque<int> temp;

    for (int i = 0; i < N; i++) {
        temp.push_back(stack[i]);

        if (isPrime(stack[i])) {
            // Only reverse if at least K elements are present
            if (temp.size() >= K) {
                reverse(temp.end() - K, temp.end());
            }
        }
    }
    while (!temp.empty()) {
        result.push_back(temp.front());
        temp.pop_front();
    }
    for (int x : result) cout << x << " ";
    cout << endl;

    return 0;
}
```

➔ " Problem 5: The Buried Keys "

CODE :-

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

int main() {
    int n;
    cin >> n;
    queue<int> q;
    stack<int> s;
```

```
    for (int i = 0; i < n; i++) {
        int key;
        cin >> key;
        if (key % 2 == 0)
            s.push(key);
        else
            q.push(key);
    }

    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }

    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }

    return 0;
}
```

****