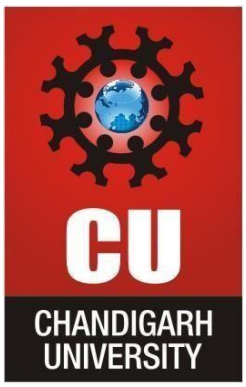

Winter Winning Camp

ASSIGNMENT – 4

Student Name: Ishan Jain

UID: 22BCS13653

Date of Submission: 24/12/2024

Branch: BE-CSE

Section/Group: 615-B

DAY – 4

```
#include <iostream>
```

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

// Very Easy Questions

// Q1: MinStack

```
class MinStack {
private:     stack<int> s;
stack<int> minStack;

public:
   MinStack() {}

   void push(int val) {
      s.push(val);         if (minStack.empty() || val <=
minStack.top()) {             minStack.push(val);
      }
   }
```

```
    void pop() {        if (s.top() ==
minStack.top()) {
minStack.pop();
        }
        s.pop();
    }

    int top() {
return s.top();
    }

    int getMin() {
return minStack.top();
    }
};
```

// Q2: First Unique Character in a String

```
int firstUniqChar(string s) {
unordered_map<char, int> count;
    for (char c : s) {
count[c]++;
    }     for (int i = 0; i < s.size();
i++) {        if (count[s[i]] == 1)
return i;
    }
return -1;
}
```

// Q3: Simple Text Editor

```
class TextEditor {
private:     string text;

stack<string> history;


public:

   TextEditor() : text("") {}


   void append(string w) {

history.push(text);         text

+= w;

   }


   void deleteLast(int k) {

history.push(text);         text =

text.substr(0, text.size() - k);

   }


   char print(int k) {

return text[k - 1];

   }


   void undo() {         if

(!history.empty()) {

text = history.top();

history.pop();

        }

   }

};
```

// Q4: Implement Queue using Stacks

```
class MyQueue { private:
stack<int> input, output;

public:
   MyQueue() {}

   void push(int x) {
input.push(x);
   }

   int pop() {        if
(output.empty()) {            while
(!input.empty()) {
output.push(input.top());
input.pop();
          }
       }        int val =
output.top();
output.pop();        return
val;
   }

   int peek() {        if
(output.empty()) {            while
(!input.empty()) {
```

```
output.push(input.top());
input.pop();
            }
        }
        return output.top();
    }

    bool empty() {         return input.empty()
&& output.empty();
    }
};
```

// Q5: Evaluate Reverse Polish Notation

```
int evalRPN(vector<string>& tokens) {     stack<int> s;     for
(const string& token : tokens) {         if (isdigit(token[0]) ||
(token.size() > 1 && token[0] == '-')) {
            s.push(stoi(token));         } else {
int b = s.top(); s.pop();             int a =
s.top(); s.pop();             if (token == "+")
s.push(a + b);             else if (token == "-")
s.push(a - b);             else if (token == "*")
s.push(a * b);             else if (token == "/")
s.push(a / b);
        }     }
return s.top();
}
```

// Q6: Valid Parentheses

```
bool isValid(string s) {
stack<char> stk;     for
(char c : s) {
       if (c == '(' || c == '{' || c == '[') {             stk.push(c);         } else {             if
(stk.empty()) return false;             char top = stk.top();             stk.pop();             if ((c
== ')' && top != '(') || (c == '}' && top != '{') || (c == ']' && top != '[')) {
return false;
           }
       }
   }
   return stk.empty();
}
```

// Q7: Number of Students Unable to Eat Lunch int

```
countStudentsUnableToEat(vector<int>& students, vector<int>& sandwiches) {
queue<int> q;     for (int student : students) {
       q.push(student);
}     int i = 0;
   while (!q.empty() && i < sandwiches.size()) {
if (q.front() == sandwiches[i]) {
           q.pop();
i++;         }
else {
           q.push(q.front());
           q.pop();
}
       if (q.size() == students.size()) return q.size();
```

```
    }     return
q.size();
}
```

// Q8: Trapping Rain Water

```
int trap(vector<int>& height) {     int n
= height.size();     if (n == 0) return 0;
vector<int> left(n), right(n);     left[0]
= height[0];     for (int i ```cpp = 1; i <
n; i++) {         left[i] = max(left[i - 1],
height[i]);
    }     right[n - 1] = height[n - 1];     for
(int i = n - 2; i >= 0; i--) {         right[i] =
max(right[i + 1], height[i]);
    }
    int waterTrapped = 0;     for (int i = 0; i < n; i++) {
waterTrapped += min(left[i], right[i]) - height[i];
    }
    return waterTrapped;
}
```

// Medium Questions

// Q9: Next Greater Element II

```
vector<int> nextGreaterElements(vector<int>& nums) {
int n = nums.size();
```

```
    vector<int> result(n, -1);     stack<int> s;     for (int i = 0;
i < 2 * n; i++) {         while (!s.empty() && nums[s.top()] <
nums[i % n]) {             result[s.top()] = nums[i % n];
            s.pop();         }
if (i < n) s.push(i);
    }     return
result;
}

// Q10: Reverse a Queue using Recursion void
reverseQueue(queue<int>& q) {
    if (q.empty()) return;
int front = q.front();
    q.pop();
reverseQueue(q);
    q.push(front);
}
```

// Q11: Score of Parentheses

```
int scoreOfParentheses(string s) {     stack<int> stk;
for (char c : s) {         if (c == '(') {             stk.push(-
1); // Use -1 to indicate a new level
        } else {             int score =
0;             while (stk.top() != -
1) {                 score +=
stk.top();                 stk.pop();
            }
```

```
        stk.pop(); // Remove the -1            stk.push(score == 0 ? 1 : 2 * score); // If
score is 0, it means we have a pair ()
      }    }    int totalScore
= 0;    while (!stk.empty())
{       totalScore +=
stk.top();       stk.pop();
  }    return
totalScore;
}
```

// Q12: Longest Increasing Subsequence II

```
int lengthOfLIS(vector<int>& nums, int k) {    int n =
nums.size();    vector<int> dp(n, 1);    for (int i = 0; i < n;
i++) {       for (int j = 0; j < i; j++) {           if (nums[i] >
nums[j] && nums[i] - nums[j] <= k) {               dp[i] =
max(dp[i], dp[j] + 1);
        }
      }
  }
  return *max_element(dp.begin(), dp.end());
}
```

// Q13: Dinner Plates Stacks

```
class DinnerPlates { private:    vector<stack<int>> stacks;
priority_queue<int, vector<int>, greater<int>> availableStacks;
int capacity;

public:
```

```
    DinnerPlates(int capacity) : capacity(capacity) {}

    void push(int val) {        if (availableStacks.empty() ||
stacks[availableStacks.top()].size() == capacity) {
stacks.push_back(stack<int>());            availableStacks.push(stacks.size() - 1);
        }
        stacks[availableStacks.top()].push(val);        if
(stacks[availableStacks.top()].size() == capacity) {
availableStacks.pop();
        }
    }

    int pop() {        if (stacks.empty()) return -1;
int index = stacks.size() - 1;        while (index >= 0
&& stacks[index].empty()) {
            index--;
        }        if (index < 0) return -1;        int val =
stacks[index].top();        stacks[index].pop();        if
(stacks[index].empty() && index == stacks.size() - 1) {
stacks.pop_back();

        }
return val;
    }

    int popAtStack(int index) {        if (index >= stacks.size() ||
stacks[index].empty()) return -1;        int val =
stacks[index].top();        stacks[index].pop();        if
```

```
(stacks[index].empty() && index == stacks.size() - 1) {
stacks.pop_back();
        }
return val;
    }
};
```

// Q14: Sliding Window Maximum

```
vector<int> maxSlidingWindow(vector<int>& nums, int k) {
vector<int> result;     deque<int> dq;     for (int i = 0; i <
nums.size(); i++) {         while (!dq.empty() && dq.front()
<= i - k) {             dq.pop_front();
        }
        while (!dq.empty() && nums[d ```cpp
q.back()] < nums[i]) {
dq.pop_back();
        }
        dq.push_back(i);
if (i >= k - 1) {
result.push_back(nums[
dq.front()]);

        }     }
return result;
}
```

// Q15: Flood Fill Algorithm

```
void floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {
int oldColor = image[sr][sc];     if (oldColor == newColor) return;     int
rows = image.size(), cols = image[0].size();     queue<pair<int, int>> q;
   q.push({sr, sc});
image[sr][sc] = newColor;


   while (!q.empty()) {
auto [r, c] = q.front();
      q.pop();        vector<pair<int, int>> directions = {{1, 0}, {-1, 0},
{0, 1}, {0, -1}};        for (auto [dr, dc] : directions) {            int newRow
= r + dr, newCol = c + dc;
         if (newRow >= 0 && newRow < rows && newCol >= 0 && newCol < cols &&
image[newRow][newCol] == oldColor) {               image[newRow][newCol] =
newColor;
            q.push({newRow, newCol});
         }
      }
   }
}
```

// Hard Questions

// Q16: Poisonous Plants

```
int poisonousPlants(vector<int>& p) {     int days = 0;
vector<int> stack;     while (true) {        vector<int>
newPlants;        for (int i = 0; i < p.size(); i++) {
```

```
while (!stack.empty() && p[i] > stack.back()) {
stack.pop_back();
        }            if (stack.empty()) {
newPlants.push_back(p[i]);
        }
        stack.push_back(p[i]);
      }
      if (newPlants.size() == p.size()) break;
p = newPlants;        days++;
   }    return
days;
}
```

// Q17: Longest Valid Parentheses

```
int longestValidParentheses(string s) {
stack<int> stk;    stk.push(-1);    int
maxLength = 0;    for (int i = 0; i <
s.size(); i++) {        if (s[i] == '(') {
        stk.push(i);
} else {
stk.pop();            if
(stk.empty()) {
stk.push(i);
        } else {                maxLength =
max(maxLength, i - stk.top());
        }
      }
   }
```

```
    return maxLength;
}
```

// Q18: Number of Students Unable to Eat Lunch

```
int countStudentsUnableToEat(vector<int>& students, vector<int>& sandwiches) {
queue<int> q;     for (int student : students) {
        q.push(student);
}     int i = 0;
    while (!q.empty() && i < sandwiches.size()) {
if (q.front() == sandwiches[i]) {
            q.pop();
i++;         }
else {
            q.push(q.front());
            q.pop();
}
        if (q.size() == students.size()) return q.size();
    }
    return q.size();
}
```

// Q19: Truck Tour

```
int truckTour(vector<pair<int, int>>& petrol) {
int total = 0, current = 0, start = 0;     for (int i =
0; i < petrol.size(); i++) {         total +=
petrol[i].first - petrol[i].second;         current +=
petrol[i].first - petrol[i].second;         if (current
< 0) {             start = i + 1;             current = 0;
```

```
    }   }   return total >= 0
? start : -1;
}
```

// Q20: Zuma Game

```
int findMinBalls(string board, string hand) {
unordered_map<char, int> handCount;
for (char c : hand) {       handCount[c]++;
  }
  function<int(string)> dfs = [&](string b) {
if (b.empty()) return 0;       b += '#'; // Add
a delimiter       int res = INT_MAX, count
= 0;       for (int i = 1; i < b.size(); i++) {
if (b[i] == b[i - 1]) {             count++;
        } else {
if (count >= 2) {
            b.erase(i - count, count + 1);
i = i - count; // Reset index
count = 0;
          }
        }
      }
      if (b.size() == 1) return 0; // All balls cleared
for (char c : handCount) {          if (hand ```cpp
Count[c] > 0) {            handCount[c]--;            int
needed = 3 - count; // Balls needed to clear            if
(needed > 0) {              if (handCount[c] >=
needed) {                  handCount[c] -= needed;
```

```
int result = dfs(b);                    if (result != -1) {
res = min(res, result + needed);
                    }
                    handCount[c] += needed; // Backtrack
                }               } else {
int result = dfs(b);                    if
(result != -1) {                    res =
min(res, result);
                }
            }
            handCount[c]++;
        }
    }
    return res == INT_MAX ? -1 : res;
  };    return
dfs(board);
}
```

// Q21: Maximum Water Trapped

```
int trap(vector<int>& height) {    int n
= height.size();    if (n == 0) return 0;
vector<int> left(n), right(n);    left[0]
= height[0];    for (int i = 1; i < n; i++)
{       left[i] = max(left[i - 1],
height[i]);
  }    right[n - 1] = height[n - 1];    for
(int i = n - 2; i >= 0; i--) {       right[i] =
max(right[i + 1], height[i]);
```

```
    }
    int waterTrapped = 0;     for (int i = 0; i < n; i++) {
waterTrapped += min(left[i], right[i]) - height[i];
    }
    return waterTrapped;
}
```

// Q22: Sorted GCD Pair Queries

```
vector<int> gcdPairs(vector<int>& nums, vector<int>& queries) {
vector<int> gcds;
    int n = nums.size();     for (int i = 0; i < n; i++)
{         for (int j = i + 1; j < n; j++) {
gcds.push_back(__gcd(nums[i], nums[j]));
        }
    }
    sort(gcds.begin(), gcds.end());
vector<int> result;     for (int q :
queries) {
result.push_back(gcds[q]);
    }     return
result;
}
```

// Q23: Longest Valid Parentheses

```
int longestValidParentheses(string s) {
stack<int> stk;     stk.push(-1);     int
maxLength = 0;     for (int i = 0; i <
s.size(); i++) {         if (s[i] == '(') {
```

```
stk.push(i);        } else {
stk.pop();            if (stk.empty()) {
stk.push(i);
          } else {                maxLength =
max(maxLength, i - stk.top());
          }
       }
   }     return
maxLength;
}
```

// Q24: Longest Subsequence with Constraints int

```
longestSubsequence(vector<int>& nums, int k) {     int n =
nums.size();     vector<int> dp(n, 1);     for (int i = 0; i < n;
i++) {         for (int j = 0; j < i; j++) {             if (nums[i] >
nums[j] && nums[i] - nums[j] <= k) {                dp[i] =
max(dp[i], dp[j] + 1);
          }
       }
   }
   return *max_element(dp.begin(), dp.end());
}
```

// Q25: Trapping Rain Water

```
int trap(vector<int>& height) {     int n
= height.size();     if (n == 0) return 0;
vector<int> left(n), right(n);     left[0]
= height[0];     for (int i = 1; i < n; i++)
```

```
{        left[i] = max(left[i - 1],
height[i]);
    }     right[n - 1] = height[n - 1];     for
(int i = n - 2; i >= 0; i--) {         right[i] =
max(right[i + 1], height[i]); }     int
waterTrapped = 0;     for (int i = 0; i < n;
i++) {         waterTrapped += min(left[i],
right[i]) - height[i];
    }
    return waterTrapped;
}
```