Experiment 1.1

Student Name: Ashutosh Dwivedi	**UID:** 21BCS2617
Branch: BE-CSE	**Section/ Group:** 21BCS-CC-616-A
Semester: 6th	**Date of Performance:** 19 Jan, 2024
Subject Name: AP-LAB-II	**Subject Code:** 21CSP-351

1. Aim: To Implement the concept of Arrays, Queues, and Stack and linked list

2. Objective:

- To learn the concept and different operations on Arrays, Queues, Stacks, and linked list.
- Along with this to implement its concepts with different approaches.
- To apply or implement different operation on it likely merging of two list and removing of duplicate values from sorted list.

3. Approach and Output:

Problem Statement 1: 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.

Code:

```cpp
class Solution{
    public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2)
    {
        ListNode* mptr=new ListNode(-1);
        ListNode* ans=mptr;
        while(list1!=NULL && list2!=NULL)
        {
            if(list1->val>=list2->val)
            {
                mptr->next=list2;
                mptr=list2;
                list2=list2->next;
            }
            else
            {
```

```cpp
            mptr->next=list1;
            mptr=list1;
            list1=list1->next;
        }
    }
    while(list1!=NULL)
    {
        mptr->next=list1;
        mptr=list1;
        list1=list1->next;
    }
    while(list2!=NULL)
    {
        mptr->next=list2;
        mptr=list2;
        list2=list2->next;
    }
    return ans->next;
    }
};
```
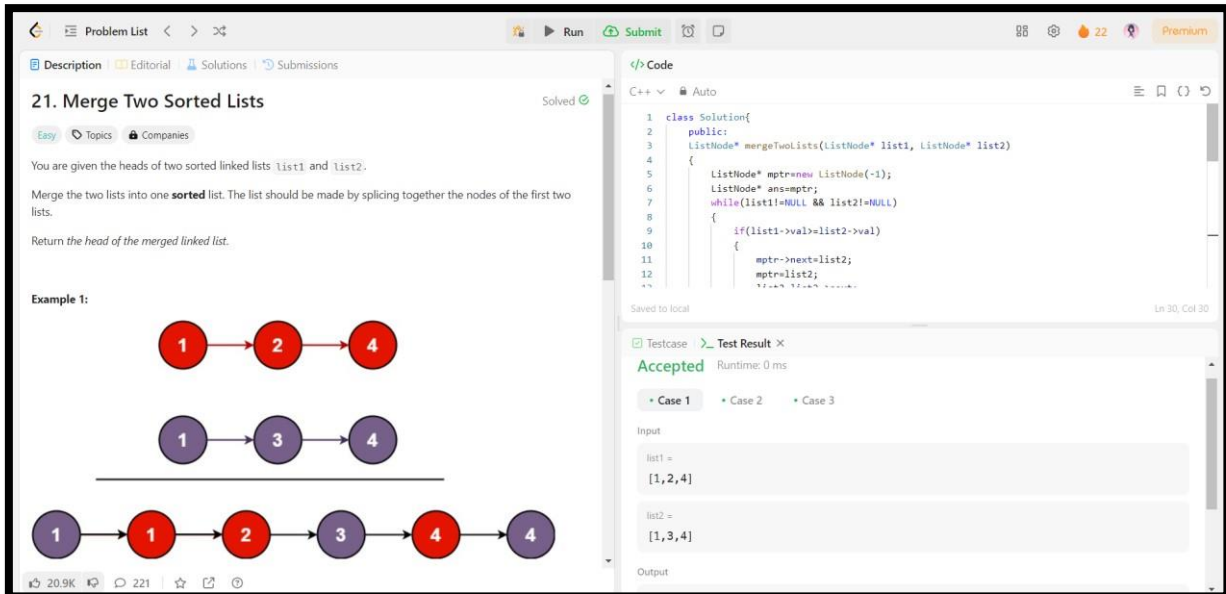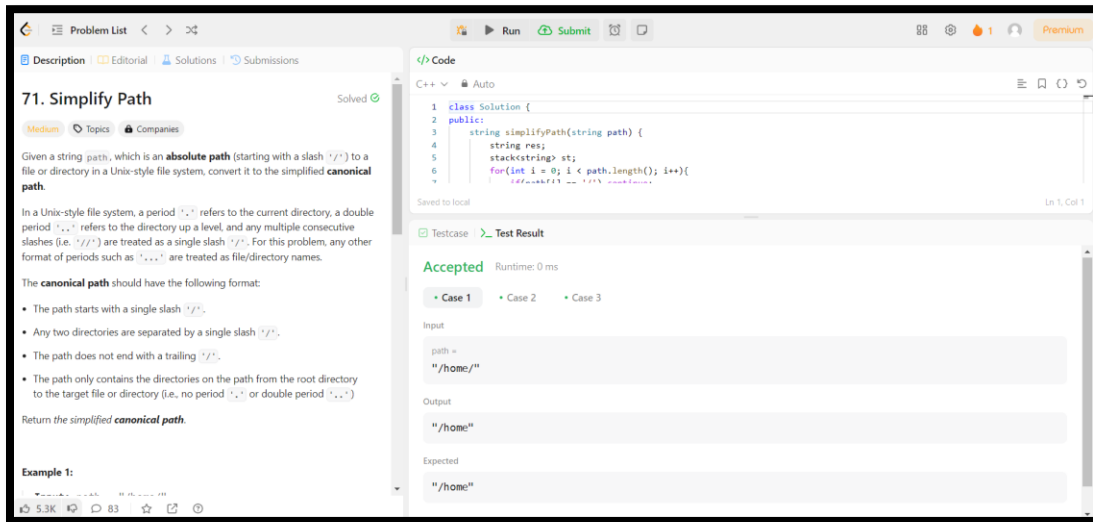
Output:

Approach:

The given problem is that to merge the two sorted list where only heads are given. To solve this problem the approach is used is that the merging process of both the linked list can be done through iterative approach of both the list by appending the smallest values and the pointer will move next ahead. If List1 is having the smaller value then it will write the list1 values else move towards the list2. If list1 is having itself the smaller value then pointer of list1 will corresponding move next to the value and will re compare its value to list 2. This approach of iterating the two linked list and merging them will create a one single list in sorted manner.

Problem Statement 2: Given a string path, which is an absolute path (starting with a slash '/') to a file or directory in a Unix-style file system, convert it to the simplified canonical path.
In a Unix-style file system, a period '.' refers to the current directory, a double period '..' refers to the directory up a level, and any multiple consecutive slashes (i.e. '//') are treated as a single slash '/'. For this problem, any other format of periods such as '...' are treated as file/directory names.

Code:

```cpp
class Solution {
public:
    string simplifyPath(string path) {
        string res;
        stack<string> st;
        for(int i = 0; i < path.length(); i++){
            if(path[i] == '/') continue;
            string temp;
            while(i < path.size() && path[i] != '/'){
                temp += path[i];
                i++;
            }
            if(temp == ".") continue;
            else if(temp == ".."){
                if(!st.empty())
                    st.pop();
            }
            else st.push(temp);
        }
        while(!st.empty())
        {
            res = '/' + st.top() + res;
            st.pop();
        }
        if(res.length() == 0) return "/";
        return res;
    }
};
```

Output:

Approach:

To convert an absolute path to its simplified canonical form in a Unix-style file system, the algorithm employs a stack to track directory components. It tokenizes the input path using slashes as delimiters, skipping empty tokens and handling ".", ".." appropriately. For each token, it either pushes it onto the stack or pops the top element in the case of "..". After processing, the algorithm constructs the simplified canonical path by joining stack elements with slashes. It ensures the resulting path starts with a slash and, if the stack is empty, returns "/". This approach efficiently handles consecutive slashes and accurately represents the simplified canonical path for the given Unix-style file system.