

Experiment 1.1

Student Name: Ashutosh Dwivedi	UID: 21BCS2617
Branch: BE-CSE	Section/ Group: 21BCS-CC-616-A
Semester: 6 th	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:

```
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
            {

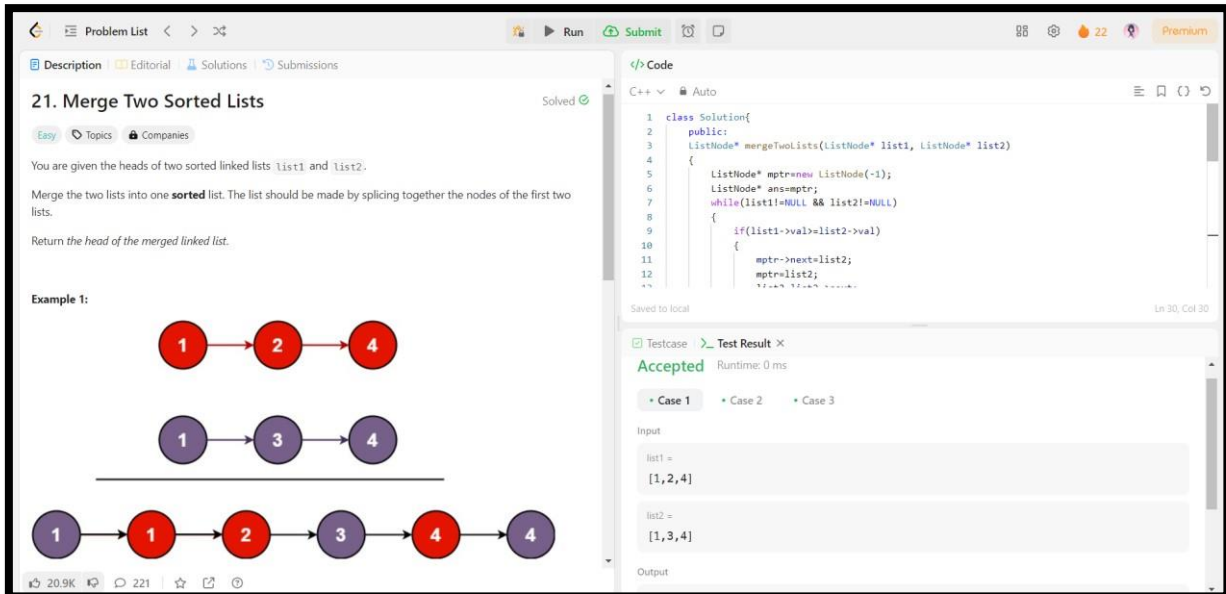
```

```

        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:



21. Merge Two Sorted Lists Solved

Easy Topics Companies

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.

Return the head of the merged linked list.

Example 1:

```

graph LR
    subgraph List1 [list1]
        direction LR
        L1_1((1)) --> L1_2((2)) --> L1_3((4))
    end
    subgraph List2 [list2]
        direction LR
        L2_1((1)) --> L2_2((3)) --> L2_3((4))
    end
    subgraph Merged [Merged List]
        direction LR
        M_1((1)) --> M_2((1)) --> M_3((2)) --> M_4((3)) --> M_5((4)) --> M_6((4))
    end

```

Code:

```

1 class Solution{
2 public:
3     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2)
4     {
5         ListNode* mptr=new ListNode(-1);
6         ListNode* ansmptr;
7         while(list1!=NULL && list2!=NULL)
8         {
9             if(list1->val>list2->val)
10            {
11                mptr->next=list2;
12                mptr=list2;
13            }
14            else
15            {
16                mptr->next=list1;
17                mptr=list1;
18            }
19            if(list1!=NULL) list1=list1->next;
20            if(list2!=NULL) list2=list2->next;
21        }
22        mptr->next=list1!=NULL?list1:list2;
23        return mptr;
24    }
25 };

```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

list1 =
[1,2,4]

list2 =
[1,3,4]

Output

Approach:

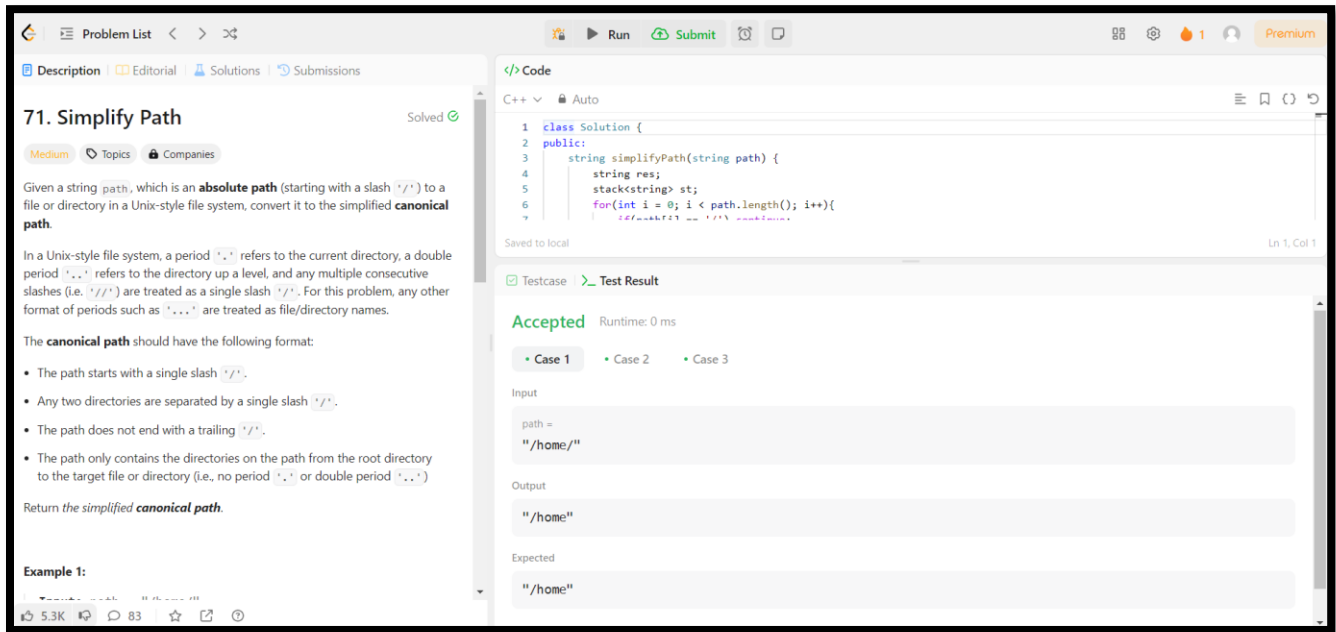
The given problem is that to merge the two sorted list where only heads are given. To solve this problem the approach is used is that the merging process of both the linked list can be done through iterative approach of both the list by appending the smallest values and the pointer will move next ahead. 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:

```
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:



The screenshot displays a coding interface for the problem "71. Simplify Path". The problem description on the left explains that the goal is to convert an absolute path (starting with '/') into its simplified canonical form. It details the rules for handling periods ('.', '..', and '..') and provides an example: "/home/" simplifies to "/home".

The code editor on the right shows a C++ solution using a stack to track directory components. The code iterates through the path, pushing valid directory names onto the stack and popping them when encountering '..'. Finally, it joins the stack elements with slashes to form the canonical path.

```

1 class Solution {
2 public:
3     string simplifyPath(string path) {
4         string res;
5         stack<string> st;
6         for(int i = 0; i < path.length(); i++){
7             if(path[i] == '/' || path[i] == '\0')
8                 continue;
9             string token = path.substr(i, path[i] == '\0' ? path.length() - i : path[i] == '/' ? i - i : 1);
10            if(token != "." && token != "..")
11                st.push(token);
12            else if(token == ".." && !st.empty())
13                st.pop();
14        }
15        if(st.empty())
16            return "/";
17        res = st.top();
18        while(st.size() > 1)
19            res = st.top() + "/" + res;
20        st.pop();
21        return res;
22    }
23 };

```

The test results section shows that the solution is "Accepted" with a runtime of 0 ms. The input path is "/home/" and the output is "/home".

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.