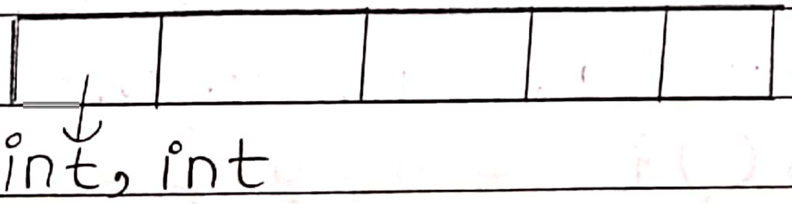
28/04/2023

Q1 Design a stack that supports push, pop, top, and retrieve minimum element in O(1) time.

Here we can use the vector.



vector <pair <int, int>> will be used where first element will be the element inserted and second element will the minimum element till now. Minimum element → V.back().second.

↳ Ans (minElement)

Code

```
class   MinStack {
      public:
            vector <pair <int, int>> st;
            MinStack(){

            }
            void push (int val) {
               // Empty case
               // Both values same in this case
               if (st.empty()) {
                     pair <int,int> p = make_pair
                                          (val, val);
                     st.push_back(p);   same
               }
               //Not empty case
               else {
                  pair <int, int> p;
                  p.first = val;
                  //Minimum till now to be inserted
                  p.second = min (val, st.back().
                                          second);
                  st.push_back(p);              last
            }                                   elemnet of
          }                                     vector
          void pop(){ //Simply pop
              st.pop_back();
          }
          int top(){// First element in pair of
              return st.back().first;  last
          }                          element in
                                     vector
```

```
    int getMin() {
        //Second element of pair stored at last
        in vector will be minimum
        return st.back().second;
    }
};
```

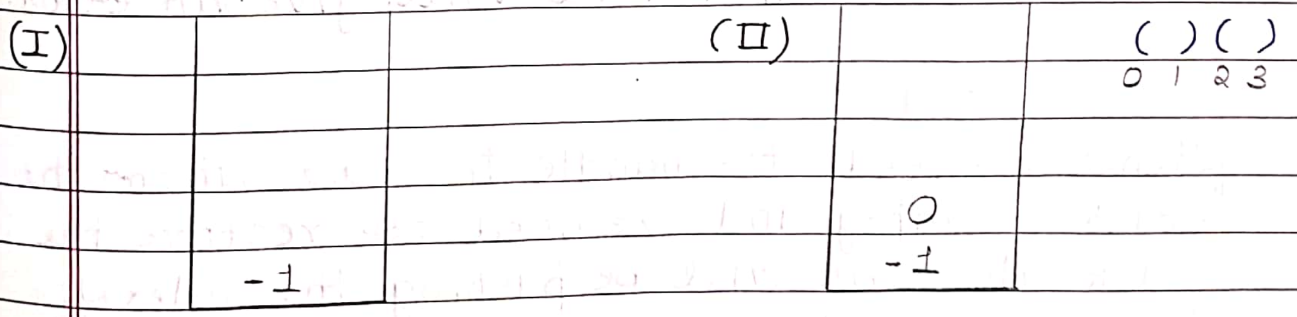
Note → back function used to find the last element of the vector.

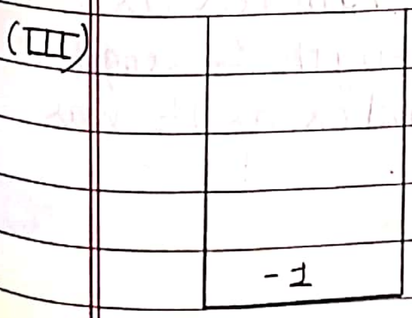
Q2 Longest valid paranthesis.

i/p →)()())

o/p → 4 → $\underbrace{()()}_{\text{Length}}$

Incase when the string given is empty, then we have to return 0. Initially insert -1 in stack. Whenever we encounter the open bracket, store its index in the stack.



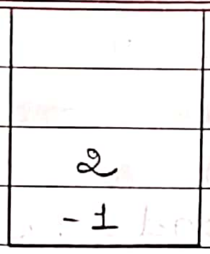


) → In the case of closing bracket pop the index of opening bracket.

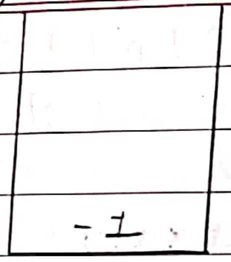
Find length = 1 − (−1) = 2

↳ index of closing

(IV)



(V)

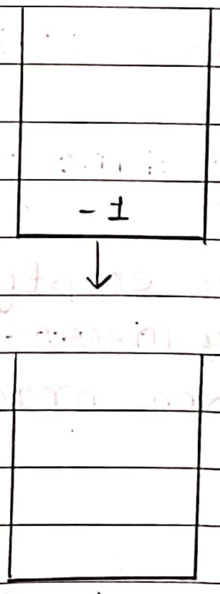


Length = 3 - (-1) = 4

Hence return

Unhappy case

)) → i/p



First closing bracket was encountered. So simply pop.

Here while finding the length we will be using s.top() but here code will give an error.

↳ empty stack

Hence we need to handle the case when the stack is empty and we need to restore the stack. Here we will be pushing the index of the bracket which needs to be ignored as it won't be considered in the length & length will be calculated after that index as it was invalid case.

Code

```
int longestValidParantheses (string s){
      //Create stack
      stack <int>st;
      //Initially add/push -1 in stack
      St.push (-1);
      int maxLen = 0;
      //Traverse the string
      for (int i=0; i< s.length (); i++){
            char ch = s[i];
            //Opening bracket
            if (ch == '(') {
                  st.push(i); //Push index
            }
            else{ //Closing bracket → simply pop
            st.pop();
            //Stack empty?
            if (st.empty()){                 Important
                        st.push (i);         condition &
            }                                addition.
            else{
                  //Not empty → calculate length
                  int len = i - st.top();
                  maxLen = max (len, maxLen);
            }                  ↪ maximum length we
        }                          need to find.
    }
    return maxLen;
}
```