Valid Parenthesis

Approach :

Use a stack because for every opening bracket there must be a closing bracket next. Idea is to push into stack if open brackets are e

Edge cases to consider :

1.]() if the initial bracket is a closing bracket. In this case the stack will be empty so before checking the top elem a check need to b
2. [( ]) - second case to consider is whether the top of stack does not match then we return false.
3. [[( )] - All the other brackets will be popped from the stack except the first opening bracket as a closing bracket for it is not founc

Code :

```
bool isValidParenthesis(string expression)
{
    // Write your code here.
    stack<int> s;
    for(int i=0;i<expression.size();i++)
    {
        if(expression[i]=='(' || expression[i]=='{' || expression[i]=='[')
        {
            s.push(expression[i]);
        } else {
          if (s.empty())
            return false;
          char c = s.top();
          s.pop();
          if ((expression[i] == ')' && c == '(') || (expression[i] == '}' && c=='{') || (expression[i]=']' && c=='['))
          {
              continue;
          }
          else{
              return false;
          }
        }
    }
    if(s.empty())
        return true;
    return false;


}
```

- Time Complexity : O(N)
- Space Complexity : O(N) stack