05/02/24

Q11. Given a balanced parentheses string s, return the score of the string. The score of a balanced parentheses string is based on the following rule:

- "()" has score 1.
- AB has score A+B, where A and B are balanced parentheses strings.
- (A) has score 2*A, where A is a balanced parentheses string.

Example 1:
Input: s = "()"
Output: 1
Example 2:
Input: s = "(())"
Output: 2

Constraints:
- 2 <= s.length <= 50
- s consists of only '(' and ')'.
- s is a balanced parentheses string.

```
#include <stdio.h>
#include <string.h>
int scoreOfParentheses(char *s)
{
  int stack[50];
  int top = -1;
  int score=0;
  for(int i=0; s[i]!='\0'; i++)
  {
    if(s[i]=='(')
    {
      stack[++top]=score;
      score=0;
```

```
    }
    else
    {
      score = stack[top--] + (score == 0 ? 1 : 2 * score);
    }
  }
  return score;
}
int main()
{
  printf("%d\n", scoreOfParentheses("()"));
  printf("%d\n", scoreOfParentheses("(())"));
  printf("%d\n", scoreOfParentheses("()()"));
  return 0;
}
```

NL
19/2/24

19/02/24

Write a program to construct, traverse (inorder, postorder, preorder) and display.

```
typedef struct BST{
int data;
struct BST *left;
struct BST *right;
}node;
node * create()
{
  node *temp;
  printf("Enter the data :");
  temp=(node*)malloc(sizeof(node));
  scanf("%d",&temp->data);
  temp->left=temp->right=NULL;
  return temp;
}
void insert(node * root, node *temp)
{
  if(temp->data<root->data)
  {
    if(root->left!=NULL)
      insert(root->left,temp);
    else
      root->left=temp;
  }
  if(temp->data>root->data)
  {
    if(root->right!=NULL)
      insert(root->right,temp)
    else
      root->right=temp;
  }
}
```

```
void inorder(node *root)
{
  if(root != NULL)
  {
    inorder(root->left)
    printf("%d", root->data);
    inorder(root->right);
  }
}
void postorder(node *root)
{
  if(root != NULL)
  {
    postorder(root->left);
    printf("%d", root->data);
    postorder(root->right);
  }
}
void preorder(node* root)
{
  if(root != NULL)
  {
    printf("%d", root->data);
    preorder(root->left);
    preorder(root->right);
  }
}
void main()
{
  root = NULL;
  root = insert(root, 10);
  root = insert(root, 20);
  root = insert(root, 30);
  root = insert(root, 40);
```

```
root = insert(root, 50);
root = insert(root, 60);
root = insert(root, 70);
printf("Insertion successful \n");
inorder(root);
printf("\n");
preorder(root);
printf("\n");
postorder(root);
printf("\n");
}
```

O/P:

Insertion successful

10 20 30 40 50 60 70

10 50 20 60 70 30 40

50 70 60 40 30 20 10

```
1. Preorder
2. Inorder
3. Postorder
4. Exit
Choice: 1
8 3 1 6 4 7 10 14 13
Enter choice: 2
1 3 4 6 7 8 10 13 14
Enter choice: 3
1 4 7 6 3 13 14 10 8
Enter choice: 4

Process returned 0 (0x0)   execution time : 14.294 s
Press any key to continue.
```

19/02/24

→ Delete the middle node of a linked list:-

```
struct ListNode *deleteMiddle(struct ListNode* head)
{
  struct ListNode* ptr=head;
  struct ListNode* freeptr =NULL;
  int count =0;
  int n=0;
  if(head= NULL)
  {
    return head;
  }
  else if(head ->next == NULL)
  {
    free(head);
    return NULL;
  }
  else
  {
    while(ptr!=NULL)
    {
      count++;
      ptr = ptr ->next;
    }
    ptr = head;
    while( n! =count/2)
    {
      freeptr=ptr;
      ptr = ptr-> next;
      n++;
    }
    freeptr ->next=ptr ->next;
    free (ptr);
  }
} return (head);
```

← All Submissions

Accepted

user0435PL submitted at Feb 26, 2024 09:25

Editorial | Solution

Runtime

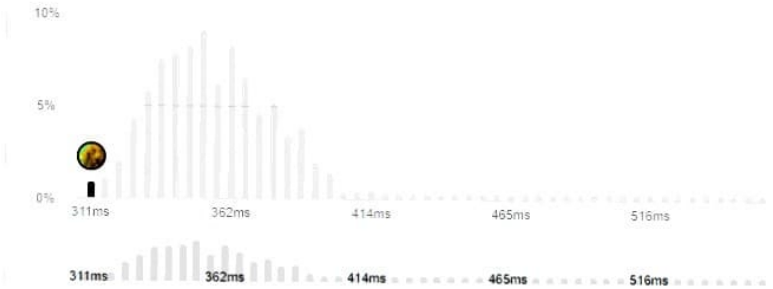
313 ms

Beats 99.42% of users with C

Memory

77.78 MB

Beats 70.12% of users with C



Code C

```c
// Define the structure for a singly linked list node
struct Node {
    int data;
    struct Node* next;
};

// Function to delete the middle node
struct Node* deleteMiddle(struct Node* head) {
```

View more

More challenges

- 19. Remove Nth Node From End of List
- 143. Reorder List
- 203. Remove Linked List Elements

C Auto

```c
    // Initialize slow_ptr and fast_ptr
    struct Node* slow_ptr = head;
    struct Node* fast_ptr = head;
    struct Node* prev;

    // Move fast_ptr by 2 steps and slow_ptr by 1 step
    while (fast_ptr != NULL && fast_ptr->next != NULL) {
        fast_ptr = fast_ptr->next->next;
        prev = slow_ptr;
        slow_ptr = slow_ptr->next;
    }

    // Delete the middle node (slow_ptr)
    prev->next = slow_ptr->next;
    free(slow_ptr);

```

Saved to local

Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 5 ms

- Case 1
- Case 2
- Case 3

Input

head =

[1,3,4,7,1,2,6]

Output

[1,3,4,1,2,6]

Expected

[1,3,4,1,2,6]

Contribute a testcase

19/02/24

→ Odd Even Linked List

```
struct ListNode* oddEvenList(struct ListNode* head)
{
    if(head==NULL || head->next==NULL)
    {
        return head;
    }
    struct ListNode* oddNode = head;
    struct ListNode* evenNode = oddNode->next;
    struct ListNode* headEven = evenNode;
    while(evenNode!=NULL && evenNode->next!=NULL)
    {
        oddNode->next = evenNode->next;
        oddNode = oddNode->next;
        evenNode->next = oddNode->next;
        evenNode = evenNode->next;
    }
    oddNode->next = headEven;
    return head;
}
```

Accepted

user0435PL submitted at Feb 26, 2024 09:33

Editorial | Solution

Runtime

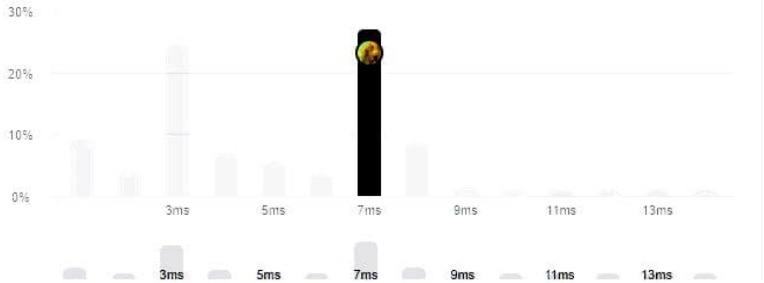
7 ms

Beats 45.53% of users with C

Memory

6.53 MB

Beats 95.61% of users with C



Code | C

```c
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
#include <stdio.h>
```

View more

More challenges

- 725. Split Linked List in Parts

Write your notes here

Code

C | Auto

```c
}

odd->next = evenHead;
return head;
}
struct ListtNode* newNode(int val)
{
struct ListtNode* node = (struct ListtNode*)malloc(sizeof(struct ListtNode));
node->val = val;
node->next = NULL;
return node;
}

```

Saved to local

Ln 35, Col 1

Testcase | Test Result

Case 1 | Case 2

head =

[1,2,3,4,5]

Source