LAB - 5

5/2/24

Q) WAP to implement doubly link list with the primitive operations

(a) Create a doubly linked list

(b) Insert a new node to the left of the node

(c) Delete the node based on a specific value

```
#include <stdio.h>
#include <stdlib.h>
struct node
{   int data;
    struct node *next,*prev;
};
struct node *start = NULL;
struct node *create_ll( struct node *);
struct node *display (struct node *);
struct node *insert_beg (struct node *);
struct node *insert_end (struct node *);
struct node *insert_before(struct node *);
struct node *insert_after (struct node *);
struct node *delete_beg (struct node *);
struct node *delete_end (struct node *);
struct node *delete_before (struct node *);
struct node *delete_after (struct node *);
struct node *delete_list (struct node *);
```

```
struct node *create_ll (struct node * start)
{
    struct node *newnode, *ptr;
    int num;
    printf ("\nEnter -1 to end\n");
    printf ("Enter a value : ");
    scanf ("%d", &num);
    while (num != -1)
    {
        if (start == NULL)
        {
            newnode = (struct node *) malloc (size of
                        (struct node));
            newnode->prev = NULL;
            newnode->data = num;
            newnode->next = NULL;
            start = newnode;
        }
        else
        {
            ptr = start;
            while (ptr->next != NULL)
            {   ptr = ptr->next
            newnode = (struct node*) malloc (size of
                        (struct node));
            newnode->data = num;
```

```
        ptr->next = newnode;
        newnode->prev = ptr;
        newnode->next = NULL;
    }
    printf("\n Enter a value : ");
    scanf("%d", &num);
}
return start;
};

struct node *display(struct node *start)
{
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
        printf("\t %d", ptr->data);
        ptr = ptr->next;
    }
    return start;
};

Struct node *insert_beg(struct node *start)
{
    struct node *newnode;
    int num;
```

```
printf ("\n Enter data to insert : ");
scanf (" %d", &num);
newnode = (struct node *) malloc (sizeof (struct node));
newnode -> data = num ;
start -> prev = newnode ;
newnode -> next = start ;
newnode -> prev = NULL ;
start = newnode ;
return start ;
};

struct node * insert_end (struct node *start)
{
  struct node *newnode, *ptr ;
  int num ;
  start ptr = start ;
  printf ("\n Enter data to insert : ") ;
  scanf ("%d", &num) ;
  newnode = (struct node *) malloc (sizeof (struct node));
  newnode -> data = num ;
  while (ptr -> next != NULL)
      ptr = ptr -> next ;
  ptr -> next = newnode ;
  newnode -> prev = ptr ;
  newnode -> next = NULL ;
  return start ;
};
```

```
struct node *insert_before (struct node *start)
{
    struct node *ptr, *newnode;
    int num, val;
    printf ("\nEnter data to insert :");
    scanf ("%d", &num);
    printf ("\nEnter value before which u want to insert:");
    scanf ("%d", &val);
    newnode = (struct node*) malloc (sizeof (struct node));
    newnode -> data = num;
    ptr = start;
    while (ptr -> data != val)
        ptr = ptr -> next
    newnode -> next = ptr;
    newnode -> prev = ptr -> prev;
    ptr -> prev -> next = newnode
    ptr -> prev = newnode;
    return start;
};

struct node *insert_after (struct node *start)
{
    struct node *ptr, *newnode;
    int num, val;
    ptr = start;
```

```
printf("\n Enter data to insert : ");
scanf("%d", &num);
printf("\n Enter a value after which u want to insert");
scanf("%d", &val);
newnode = (struct node*) malloc(sizeof(struct node*));
newnode -> data = num;
while (ptr -> data != val)
    ptr = ptr -> next;
newnode -> prev = ptr;
newnode -> next = ptr -> next;
ptr -> next -> prev = newnode;
ptr -> next = newnode;
return start;
};

struct node * delete_beg (struct node * start)
{
    struct node * start;   (ptr)
    ptr = start;
    start = start -> next;
    start -> prev = NULL;
    free(ptr);
    return start;
};
```

```
struct node *delete_end (struct node * start)
{
   struct node *ptr, *temp ;
   ptr = start ;
   while (ptr -> next != NULL)
        ptr = ptr->next ;
   temp = ptr->prev ;
   ptr->prev->next = NULL ;
   free(ptr) ;
   return start ;
};

struct node * delete_before (struct node * start)
{
   struct node *ptr, *temp ;
   int val ;
   printf ("\nEnter the value before which u to delete: ");
   scanf ("%d", &val) ;
   ptr = start ;
   while (ptr->data != val)
        ptr = ptr->next ;
   temp = ptr->prev ;
   if (temp == start)
        start = delete_beg (start) ;
```

```
else
{
    ptr->prev = temp->prev;
    temp->prev->next = ptr;
}
free(temp);
return start;
};

struct node *delete_after (struct node *start)
{
    struct node *ptr, *temp;
    int val;
    printf("enter the value after which u want to delete");
    scanf("%d", &val);
    ptr = start;
    while (ptr->data != val)
    {
        ptr = ptr->next;
    }
    temp = ptr->next;
    ptr->next = temp->next;
    temp->next->prev = ptr;
    free(temp);
    return start;
};
```

```
struct node * delete_list (struct node * start)
{
    while (start != NULL)
        start = delete_beg(start);
    return start;
};

int main ()
{
   int choice;
   printf ("*****Main Menu*****");
   printf ("1. create list \t 2. display list \t 3. insert
    beging \t 4. inser end \t 5. insert before \t 6.
    insert after \n 7. delete begining \t 8. delete end
    \t 9. delete before \t 10. delete after \t 11. delete list");
   while(1) {
     printf ("\n Enter your choice :");
     scanf ("%d", &choice);
    switch (choice)
    {
      case 1 : start = create_ll (start);
               printf ("\n doubly linked list created");
               break;
     case 2 : start = display (start);
               break;
```

```
case 3 : start = insert_beg(start);
         break;
case 4 : start = insert_end(start);
         break;
case 5 : start = insert_before(start);
         break;
case 6 : start = insert_after(start);
         break;
case 7 : start = delete_beg(start);
         break;
case 8 : start = delete_end(start);
         break;
case 9 : start = delete_before(start);
         break;
case 10 : start = delete_after(start);
         break;
case 11 : start = delete_list(start);
         printf("doubly linked list deleted");
         break;
default : exit(0);
}
}
return 0;
}
```

Leetcode

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.

```
#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;
}
```