Lab-6

* Write a program to implement doubly Link list with primitive options.
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>

struct node{
        int data;
        struct node *prev;
        struct node *next;
};

struct node *head=0, *newnode, *temp;

void create (){
        int i, n;
        printf("Enter the no. of elements: \n");
        scanf("%d", &n);

        for(i=0; i<n; i++)
        {
            newnode = (struct node *)malloc
                        (sizeof (struct node));
```

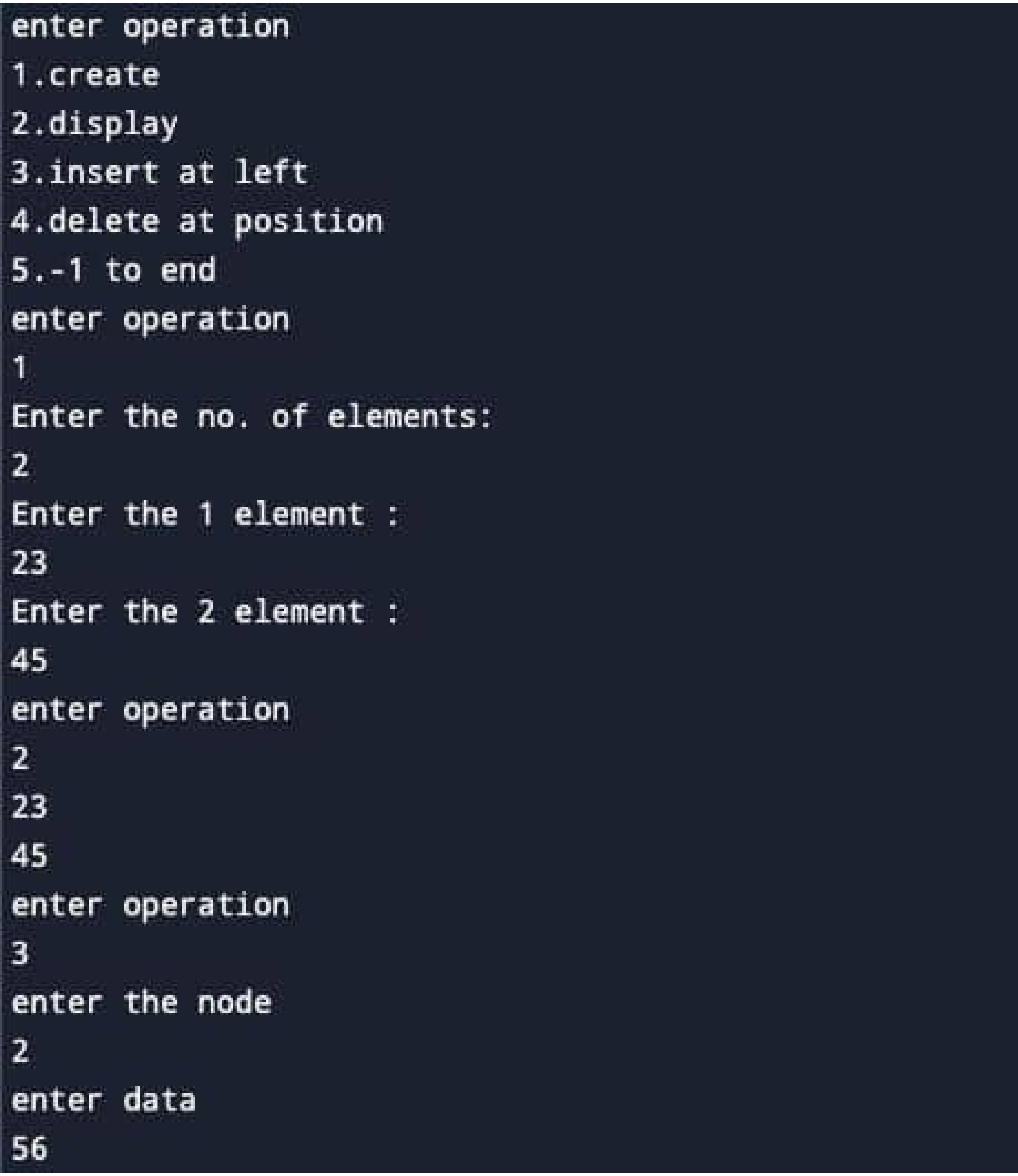
19/2/24

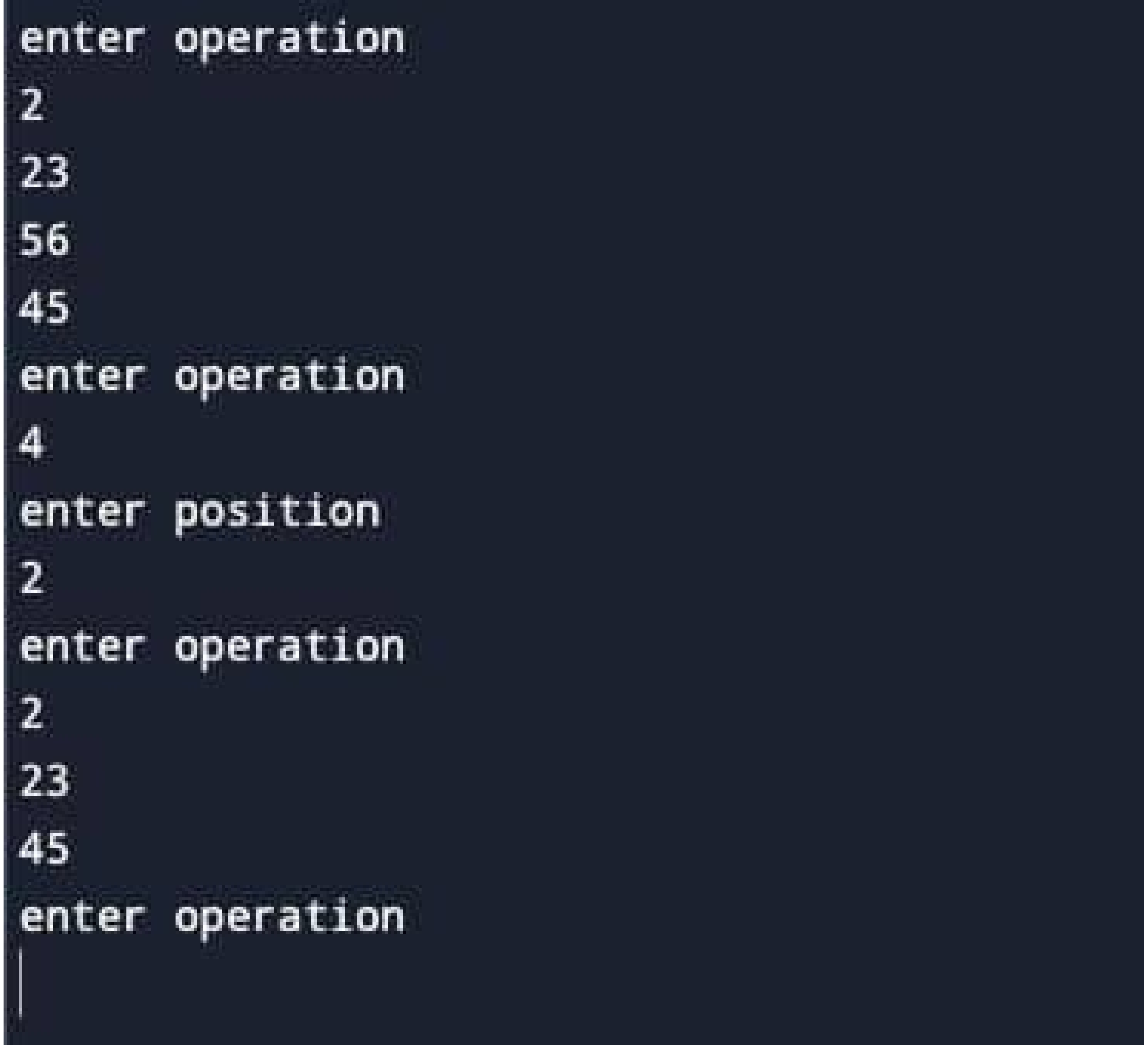
```
        printf("Enter the %d element:\n
         i+1);
        scanf("%d", &newnode ->data);
        newnode -> prev = 0;
        newnode -> next = 0;

        if(head==0)
        {
            temp = head = newnode;
        }
        else
        {
            temp -> next = newnode;
            newnode -> prev = temp;
            temp = newnode;
        }
    }
}

void display()
{
    temp=head;
    while(temp!=0)
    {
        printf("%d\n", temp -> data);
        temp = temp -> next;
    }
}
```

```
void insert_left()
{
    int node, i=1;
    printf("Enter the node\n");
    scanf("%d", &node);
    temp = head;
    if (node<1){
        printf("Invalid position\n");
    }
    else if (node==1){
        newnode=(struct node *)malloc
        (sizeof(struct node));
        printf("enter data\n");
        scanf("%d", &newnode->data);
        newnode ->prev =0;
        head -> prev = newnode;
        newnode -> next = head;
        head = newnode;
    }
    else{
        newnode = (struct node *)malloc
        (sizeof(struct node));
        printf("Enter data\n");
        scanf("%d", &newnode ->data);
        while (i<node-1){
            temp =temp -> next;
            i++;
        }
```

```
            newnode -> prev = temp;
            newnode -> next = temp -> next;
            temp -> next = newnode;
            newnode -> next -> prev = newno
        }
}

void delete_pos()
{
        int pos, i=1;
        temp = head;
        printf("Enter position\n");
        scanf("%d", &pos);
        while (i<pos){
                temp = temp -> next;
                i++;
        }
        temp -> prev -> next = temp -> next;
        temp -> next -> prev = temp -> prev;
        free(temp);
}

void main()
{
    int choice, num;
    printf("Enter operation\n 1. Create\n
            2. display\n 3. insert at left\n
            4. delete at position\n 5. -1 to e
```

```
while (1)
{
    printf("Enter operation\n");
    scanf("%d", &choice);

    if (choice == -1)
    {
        printf("completed\n");
    }
    else
    {
        switch (choice){
            case 1 : create();
                     break;

            case2 : display();
                    break ;

            case3 : insert_left();
                    break;

            case4 : delete_pos();
                    break;

            default: print("Invalid");}
    }
}
}
```


```
enter operation
1.create
2.display
3.insert at left
4.delete at position
5.-1 to end
enter operation
1
Enter the no. of elements:
2
Enter the 1 element :
23
Enter the 2 element :
45
enter operation
2
23
45
enter operation
3
enter the node
2
enter data
56
```

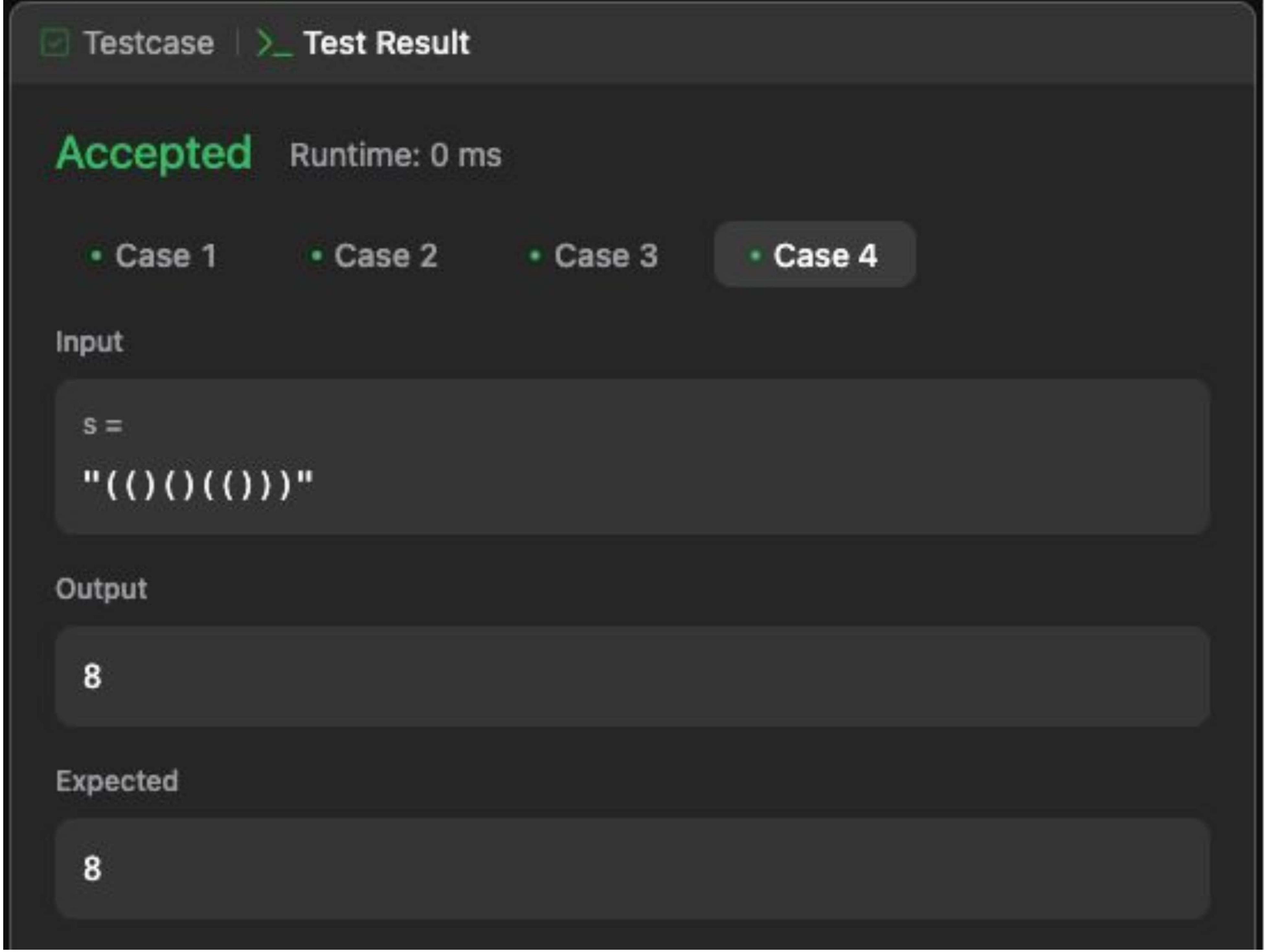

```
enter operation
2
23
56
45
enter operation
4
enter position
2
enter operation
2
23
45
enter operation
```

* LeetCode Program

⇒ Score of Parentheses

=>

```
int scoreOfParenthese (char *s) {
        int top=0, ans = 0;
        for (int i=0; i<strlen(s); i++){
                if(s[i] = '(' ){
                        top++;
                }else {
                        top--;
                        if (s[i-1]=='(' ){
                                ans += pow(2, top);
                        }
                }
        }
        return ans;
}
```



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • Case 4

Input

```
s =
"(()()(()))"
```

Output

```
8
```

Expected

```
8
```