Week 3: Singly Linked List

1. **Aim: Insert a Node at the Tail of a Linked List**
   You are given the pointer to the head node of a linked list and an integer to add to the list. Create a new node with the given integer. Insert this node at the tail of the linked list and return the head node of the linked list formed after inserting this new node. The given head pointer may be null, meaning that the initial list is empty.
   Program:

```
// Complete the insertNodeAtTail function below.

/*
 * For your reference:
 *
 * SinglyLinkedListNode {
 *     int data;
 *     SinglyLinkedListNode* next;
 * };
 *
 */


SinglyLinkedListNode* insertNodeAtTail(SinglyLinkedListNode* head, int data)
{

    if(head == NULL)
    {
        SinglyLinkedListNode *temp = new SinglyLinkedListNode(data);
        head = temp;
        return head;
    }
    SinglyLinkedListNode *cur = head;
    while(cur ->next != NULL)
    {
        cur = cur ->next;
    }
    SinglyLinkedListNode *temp = new SinglyLinkedListNode(data);
    cur ->next = temp;
    return head;
}
```

Input & Output:

Local: week3_21it068

Testcase 1 Passed 640ms

Input: Copy

```
5
141
302
164
530
474
```

Expected Output: Copy

```
141
302
164
530
474
```

Received Output: Copy

```
141
302
164
530
474
```

Testcase 2 Passed 32ms

Input: Copy

```
3
236
326
937
```

Expected Output: Copy

```
236
326
937
```

Received Output: Copy

```
236
326
937
```

2. Aim: Insert a node at the head of a linked list

Given a pointer to the head of a linked list, insert a new node before the head. The next value in the new node should point to head and the tail value should be replaced with a given value. Return a reference to the new head of the list. The head pointer given may be null meaning that the initial list is empty.

Program:

```
// Complete the insertNodeAtTail function below.

/*

 *

 For your reference:

  SinglyLinkedListNode {

     int data;

     SinglyLinkedListNode* next;

 };

 */

SinglyLinkedListNode* insertNodeAtHead(SinglyLinkedListNode* head, int data)
{
    if(head == NULL)
    {
        SinglyLinkedListNode *temp = new SinglyLinkedListNode(data);
        head = temp;
        return head;
    }
    SinglyLinkedListNode *temp = new SinglyLinkedListNode(data);
    temp ->next = head;
    head = temp;
    return head;
}
```

Input & Output:

Testcase 1 Passed 572ms

Input: Copy

```
5
383
484
392
975
321
```

Expected Output: Copy

```
321
975
392
484
383
```

Received Output: Copy

```
321
975
392
484
383
```

Testcase 2 Passed 32ms

Input: Copy

```
10
321
641
653
524
952
337
955
891
590
133
```

Expected Output: Copy

```
133
590
891
955
337
952
524
653
641
321
```

Received Output: Copy

```
133
590
891
955
337
952
524
653
641
321
```

3. Aim: Insert a node at a specific position in a linked list

Given the pointer to the head node of a linked list and an integer to insert at a certain position, create a new node with the given integer as its data attribute, insert this node at the desired position and return the head node.

A position of 0 indicates head, a position of 1 indicates one node away from the head and so on. The head pointer given may be null meaning that the initial list is empty.

Program:

```
/*
  For your reference:
  SinglyLinkedListNode {
      int data;
      SinglyLinkedListNode* next;
 };
 */
SinglyLinkedListNode* deleteNode(SinglyLinkedListNode* llist, int pos) {
    if(pos == 0)
        return llist->next;
    pos --;
    SinglyLinkedListNode *temp  = llist;
    while(pos--)
    {
        temp = temp ->next;
    }
    temp ->next  = temp ->next ->next;
    return llist;

}
```

Input & Output:

Local: week3_21it068

Testcase 1 Passed 581ms

Input: Copy

```
8
20
6
2
19
7
4
15
9
3
```

Expected Output: Copy

```
20 6 2 7 4 15 9
```

Received Output: Copy

```
20 6 2 7 4 15 9
```

Testcase 2 Passed 33ms

Input: Copy

```
4
11
9
2
9
1
```

Expected Output: Copy

```
11 2 9
```

Received Output: Copy

```
11 2 9
```

4. **Aim: Delete a Node:**
Delete the node at a given position in a linked list and return a reference to the head node. The head is at position 0. The list may be empty after you delete the node. In that case, return a null value.

Program:

```
 * For your reference:


  SinglyLinkedListNode {
      int data;
      SinglyLinkedListNode* next;
 * };
 */
SinglyLinkedListNode* deleteNode(SinglyLinkedListNode* llist, int pos) {
    if(pos == 0)
        return llist->next;
    pos --;
    SinglyLinkedListNode *temp  = llist;
      while(pos--)
    {
        temp = temp ->next;
    }
    temp ->next  = temp ->next ->next;
    return llist;
}
```

Input & Output:

Testcase 1

Input:

```
8
20
6
2
19
7
4
15
9
3
```

Expected Output:

```
20 6 2 7 4 15 9
```

Testcase 2

Input:

```
4
11
9
2
9
1
```

Expected Output:

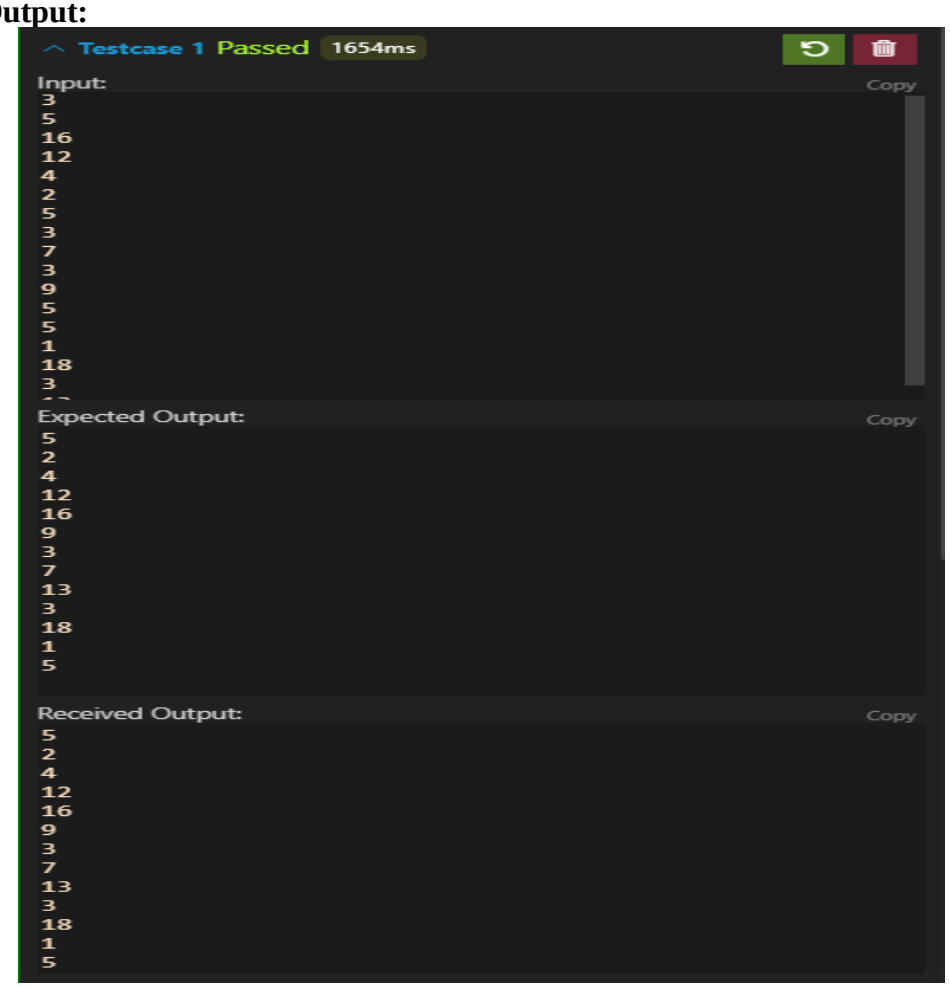
```
11 2 9
```

5. **Aim: Print in Reverse:**
Given a pointer to the head of a singly-linked list, print each data value from the reversed list. If the given list is empty, do not print anything.

Program:

```
void reversePrint(SinglyLinkedListNode* llist) {
    if(llist == NULL)
    {
        return ;
    }
    reversePrint(llist ->next);
    cout<<llist->data<<endl;
}
```

Input & Output:



Input: Copy

```
3
3
11
1
17
3
12
11
15
4
5
7
15
14
```

Expected Output: Copy

```
17
1
11
15
11
12
14
15
7
5
```

Received Output: Copy

```
17
1
11
15
11
12
14
15
7
5
```

Conclusion: From all the above programs we learned the basic operations on Singly Linked List such as

1. Insertion
2. Deletion
3. Traversing