

Week 3: Linked Lists

Lab Tutor: For the lab session, please discuss about the solution for each question in the lab. You may allocate about 30 minutes for each question. No discussion about the assignment questions and practice questions will be given.

Questions:

1. Implement the `removeNode()` function for a linked list, using the lecture diagrams and pseudo-code as a reference. The prototype of the `removeNode()` function is given below:

```
int removeNode(ListNode **ptrHead, int index);
```

The function should return 1 if the delete operation is successful and 0 otherwise. Recall that the function requires a pointer to the head pointer in order to correctly delete the first node.

Write a program to test the `removeNode()` function. It should first allow the user to create a linked list of integers by appending values to the end of the list. Next, it should allow the user to delete nodes one by one based on their indices. The program can be terminated by providing an out-of-range index to delete the node.

Sample inputs and outputs:

Enter a list of numbers, terminated by any non-digit character:

```
1
2
3
4
5
a
```

Current List: 1 2 3 4 5

Enter the index of the node to be removed: 0

After the removal operation,

Current List: 2 3 4 5

Enter the index of the node to be removed: 3

After the removal operation,

Current List: 2 3 4

Enter the index of the node to be removed: 1

After the removal operation,

Current List: 2 4

Enter the index of the node to be removed: 3

The node cannot be removed.

Current List: 2 4

2. Rewrite the node removal function as `removeNode2()` by using the `LinkedList` struct defined in the lecture materials.

```
typedef struct _linkedlist{
    ListNode *head;
    int size;
} LinkedList;
```

The prototype of `removeNode2()` is given:

```
int removeNode2(LinkedList *ll, int index);
```

Sample inputs and outputs:

Enter a list of numbers, terminated by any non-digit character:

```
1
2
3
4
5
6
a
```

Current List has 6 elements: 1 2 3 4 5 6

Enter the index of the node to be removed: 0

After the removal operation,

Current List has 5 elements: 2 3 4 5 6

Enter the index of the node to be removed: 3

After the removal operation,

Current List has 4 elements: 2 3 4 6

Enter the index of the node to be removed: 1

After the removal operation,

Current List has 3 elements: 2 4 6

Enter the index of the node to be removed: 3

The node cannot be removed.

Current List has 3 elements: 2 4 6

3. Write a function `split()` that copies the contents of a linked list into two other linked lists. The function should return 1 if the operation was successful and 0 otherwise. The function prototype is given below:

```
int split(ListNode *cur, ListNode **ptrEvenList,
          ListNode **ptrOddList);
```

The function copies nodes with even indices (0, 2, 4, etc) to `evenList` and nodes with odd indices (1, 3, 5, etc) to `oddList`. The original linked list should not be modified.

Sample inputs and outputs:

Enter a list of numbers, terminated by any non-digit character:

0 1 1 2 3 5 8 13 21 34 55 89 144 a

Before split() is called:

The original list:

Current List: 0 1 1 2 3 5 8 13 21 34 55 89 144

After split() was called:

The original list:

Current List: 0 1 1 2 3 5 8 13 21 34 55 89 144

The even list:

Current List: 0 1 3 8 21 55 144

The odd list:

Current List: 1 2 5 13 34 89

4. Write a function `duplicateReverse()` that creates a duplicate of a linked list with the nodes stored in reverse. The function prototype is given below:

```
int duplicateReverse(ListNode *cur, ListNode **ptrNewHead);
```

The function should return 1 if the operation was successful and 0 otherwise.

`newHeadPtr` should point to the first node of the reversed duplicate list.

Sample inputs and outputs:

Enter a list of numbers, terminated by any non-digit character:

0 1 1 2 3 5 8 13 21 34 55 89 144 a

Before duplicateReverse() is called:

Current List: 0 1 1 2 3 5 8 13 21 34 55 89 144

After duplicateReverse() was called:

The original list:

Current List: 0 1 1 2 3 5 8 13 21 34 55 89 144

The duplicated reverse list:

Current List: 144 89 55 34 21 13 8 5 3 2 1 1 0

Practice Questions:

1. Write a function `moveMaxToFront()` that traverses a linked list of integers at most once, then moves the node with the largest stored value to the front of the list. The function prototype is given below:

```
int moveMaxToFront(ListNode **ptrHead);
```

The function should return the original index of the node with the largest value if the operation was successful and -1 otherwise.

Sample inputs and outputs:

Enter a list of numbers, terminated by any non-digit character:
0 -5 1 2 7 a

Before `moveMaxToFront()` is called:
Current List: 0 -5 1 2 7

After `moveMaxToFront()` was called:
The original index of the node with the largest value: 4.
Current List: 7 0 -5 1 2

2. Write a C function `moveEvenItemsToBack()` that moves all the even integers to the back of the linked list.

The function prototype is given as follows:

```
void moveEvenItemsToBack(LinkedList *ll);
```

3. Write a function `concatenate()` that concatenates two linked lists. The function prototype is given below:

```
int concatenate(ListNode **ptrHead1, ListNode *head2);
```

The function should return 1 if it completes successfully and 0 otherwise. The final node of the first list should point to the first node of the second list. Your function should not create any new nodes. Be sure to handle the cases where one or more of the lists are empty.

Sample inputs and outputs:

Enter a list of numbers for head1, terminated by any non-digit character:
1 2 3 4 5 6 78 -5 12 a

Enter a list of numbers for head2, terminated by any non-digit character:

1 a

Before concatenate() is called:

Head 1:

Current List: 1 2 3 4 5 6 78 -5 12

Head 2:

Current List: 1

After concatenate() was called:

Head 1:

Current List: 1 2 3 4 5 6 78 -5 12 1

Head 2:

Current List: 1