

**CE1007/CZ1007 – DATA
STRUCTURES****Assignment Questions: Linked List****Information:**

Program templates for questions 1-4 are given as separated files (**Q1_template_LL.c**, **Q2_template_LL.c**, **Q3_template_LL.c**, and **Q4_template_LL.c**). You **must use** them to implement your functions. The program contains a *main()* function, which includes a switch statement to execute different functions that you should implement. Each function can be called multiple times depending on the user's choice.

You need to submit your code to the Automated Programming Submission and Assessment System (APAS) for testing. Deadline for program submission: **April 3, 2020 (Friday) 11.59 pm**.

Assignment Questions (1-4)

1. Write a C function `moveOddItemsToBack()` that moves all the odd integers to the back of the linked list.

The function prototype is given as follows:

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

Some sample inputs and outputs sessions are given below:

If the linked list is **2 3 4 7 15 18**:

The resulting Linked List after moving odd integers to the back of the Linked List is: **2 4 18 3 7 15**

If the linked list is **2 7 18 3 4 15**:

The resulting Linked List after moving odd integers to the back of the Linked List is: **2 18 4 7 3 15**

If the current linked list is **1 3 5**:

The resulting Linked List after moving odd integers to the back of the Linked List is: **1 3 5**

If the current linked list is **2 4 6**:

The resulting Linked List after moving odd integers to the back of the Linked List is: **2 4 6**

2. Write a C function `alternateMergeLinkedList()` which inserts nodes of the second list into alternate positions of the first list. The nodes of the second list should only be inserted when there are alternate positions available in the first list.

The function prototype is given as follows:

```
void alternateMergeLinkedList(LinkedList *ll1, LinkedList *ll2);
```

For an example, assume that given two linked lists are `LinkedList1` and `LinkedList2`:

`LinkedList1: 1 2 3`

`LinkedList2: 4 5 6 7`

The resulting linked lists are:

`LinkedList1: 1 4 2 5 3 6`

`LinkedList2: 7`

The second list should become empty when the first list is larger than the second list. For an example, assume that given two linked lists are LinkedList1 and LinkedList2:

LinkedList1: **1 5 7 3 9 11**
 LinkedList2: **6 10 2 4**

The resulting Linked Lists are:

LinkedList1: **1 6 5 10 7 2 3 4 9 11**
 LinkedList2: **empty**

A sample input and output session is given below

(if the current linked lists are Linked list 1: 1 2 3 and Linked list 2: 4 5 6 7):

```
Linked list 1: 1 2 3
Linked list 2: 4 5 6 7
```

```
Please input your choice(1/2/3/0): 3
Linked list 1: 1 4 2 5 3 6
Linked list 2: 7
```

3. **(recursiveReverse)** Write a C function recursiveReverse() that recursively reverses the given linked list by changing its next pointer and its head pointer.

The function prototype is given as follows:

```
void recursiveReverse(ListNode **ptrHead);
```

For example, if the linked list is **(1, 2, 3, 4, 5)**, the resulting linked list will be **(5, 4, 3, 2, 1)**.

A sample input and output session is given below:

```
1: Insert an integer to the linked list:
2: Reversed Linked List:
0: Quit:
```

```
Please input your choice(1/2/0): 1
Input an integer that you want to add to the linked list: 1
The resulting Linked List is: 1
Please input your choice(1/2/0): 1
Input an integer that you want to add to the linked list: 2
The resulting Linked List is: 1 2
Please input your choice(1/2/0): 1
Input an integer that you want to add to the linked list: 3
The resulting Linked List is: 1 2 3
Please input your choice(1/2/0): 1
Input an integer that you want to add to the linked list: 4
The resulting Linked List is: 1 2 3 4
Please input your choice(1/2/0): 1
Input an integer that you want to add to the linked list: 5
The resulting Linked List is: 1 2 3 4 5

Please input your choice(1/2/0): 2
The resulting Linked List after reversing its elements is: 5 4 3 2 1

Please input your choice(1/2/0): 0
```

4. Write a C function frontBackSplitLinkedList() that splits the singly linked list into two sublists – one for the front half, and one for the back half. If the number of elements is odd, the extra element should go into the front list. The frontBackSplitLinkedList() prints two lists which contains frontList and backList.

The function prototypes are given as follows:

```
void frontBackSplitLinkedList(LinkedList *ll, LinkedList  
*resultFrontList, LinkedList *resultBackList);
```

For example, assume that given linked list is: **2 3 5 6 7**

The resulting Linked Lists, frontList and backList are:

frontList: **2 3 5**

backList: **6 7**

A sample input and output session is given below:

```
1: Insert an integer to the linked list:  
2: Print the linked list:  
3: Split the linked list into two linked lists, frontList and backList:  
0: Quit:
```

```
Please input your choice(1/2/3/0): 1  
Input an integer that you want to add to the linked list: 2  
The resulting linked list is: 2
```

```
Please input your choice(1/2/3/0): 1  
Input an integer that you want to add to the linked list: 3  
The resulting linked list is: 2 3
```

```
Please input your choice(1/2/3/0): 1  
Input an integer that you want to add to the linked list: 5  
The resulting linked list is: 2 3 5
```

```
Please input your choice(1/2/3/0): 1  
Input an integer that you want to add to the linked list: 6  
The resulting linked list is: 2 3 5 6
```

```
Please input your choice(1/2/3/0): 1  
Input an integer that you want to add to the linked list: 7  
The resulting linked list is: 2 3 5 6 7
```

```
Please input your choice(1/2/3/0): 2  
The resulting linked list is: 2 3 5 6 7
```

```
Please input your choice(1/2/3/0): 3  
The resulting linked lists after splitting the given linked list are:  
Front linked list: 2 3 5  
Back linked list: 6 7
```