ASSIGNMENT 1(Total 100 marks)

A. PROBLEMS (Total = 100 marks)

Program templates for questions 1-4 are given in the Appendix. 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 may be called multiple times depending on the user's choice.

1. Write a C function insertSortedLinkedList() that adds a new node to an existing linked list, maintains the list values in ascending sorted order. Note that the existing list is already in ascending sorted order.

For an example, given the existing list of numbers stored in a linked list:
2 3 5 7 9

Calling insertSortedLinkedList() with a value of 6 will result in the following linked list:
2 3 5 6 7 9

Linked lists may contain duplicate values; in this case, you should add the new value at the smallest possible index position.

The function should return the index position where the new item was added; if the function could not complete successfully, it should return a value of -1. This return value must be displayed on screen as follows:

The value 6 was added at index 3

The function prototype is given as follows:
int insertSortedLinkedList(LinkedList *ll, int item);

Your function should print the contents of the linked list after it has been created. This function may be called multiple times for each time your program is run.

Some sample inputs and outputs are given as follows:

```
1: Insert an integer to the sorted linked list:
2: Print the index of most recent input value:
3: Print sorted linked list:
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: 7
The resulting linked list is: 2 3 5 7

Please input your choice(1/2/3/0): 1
Input an integer that you want to add to the linked list: 9
The resulting linked list is: 2 3 5 7 9

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 7 9

Please input your choice(1/2/3/0):2
The value 6 was added at index 3

Please input your choice(1/2/3/0):3
The resulting linked list is: 2 3 5 6 7 9
```

(20 marks)

2. Write a function sortedMergeLinkedList() that takes two lists, each of which is sorted in ascending order, and merges two lists together into one list that is also in ascending order. sortedMergeLinkedList() should return the new list.

Write an additional C function insertSortedLinkedList() in order to create two sorted linked lists in ascending order which are used for merging.

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

LinkedList1: **2 4 6**
LinkedList2: **1 2 3**

The resulting Linked List is:
1 2 2 3 4 6

The function prototypes are given as follows:
void sortedMergeLinkedList(LinkedList *ll1, LinkedList *ll2, LinkedList *resultMergedList);
int insertSortedLinkedList(LinkedList *ll, int item);

A sample input and output session is given below (if the current linked lists are Linked list 1: 2 4 6 and Linked list 2: 1 2 3):

```
Linked list 1: 2 4 6

Linked list 2: 1 2 3

Please input your choice(1/2/3/0): 3
The resulting linked list is: 1 2 2 3 4 6
```

(30 marks)

3. Write a C function createQueueFromLinkedList() to create a queue (linked-list-based) by enqueuing all integers storing in the linked list. The first node of the linked list is enqueued first, and then the 2nd node, and so on. Remember to empty the queue in the beginning if the queue is not empty.

After the queue is built, write another C function removeEvenValues() to remove all even integers in the queue. Note that you should only use enqueue() or dequeue() when you add or remove integers from queues. You are not allowed to directly access the underlined linked list of the queue. (Hint: you may define an auxiliary queue inside the function to help for the task.)

You should also write another three C functions, enqueue(), dequeue(), and isEmptyQueue().

A sample input and output session is given below (if the current linked list is 1 2 3 4 5):

The resulting Queue is: **1 2 3 4 5**

The resulting Queue after removing *even* integers is: **1 3 5**

The function prototypes are given as follows:
void createQueueFromLinkedList(LinkedList *ll, Queue *q);
void removeEvenValues(Queue *q);
void enqueue(Queue *q, int item);
int dequeue(Queue *q);
int isEmptyQueue(Queue *q);

A sample input and output session is given below (if the current linked list is 1 2 3 4 5):

```
The resulting linked list is: 1 2 3 4 5

Please input your choice(1/2/3/0): 2
The resulting queue is: 1 2 3 4 5

Please input your choice(1/2/3/0): 3
The resulting queue after removing even integers is: 1 3 5
```

(30 marks)

4. Write a C function frontBackSplitLinkedList() that splits 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 in to the front list. The frontBackSplitLinkedList() prints two lists which contains frontList and backList.

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**

The function prototypes are given as follows:
void frontBackSplitLinkedList(LinkedList* ll, LinkedList *resultFrontList, LinkedList *resultBackList);

A sample input and output session is given below (if the current 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
```

(20 marks)

B. SUBMISSION

I. Deadline for program submission: **November 13th, 2015 (Friday) 5 pm**. You are required to submit the report (hardcopy and softcopy) and source program.

II. Penalty on late submission: Deduction of 10% of the total mark each day (working day) after the submission deadline.

III. **Report (hardcopy) Submission** - Your report should contain the following:

1. **Program listings**: Give the program listing with comments. The source program should be presented **with proper indentation** and **appropriate comments**: **Indentation** - code indentation is to improve the readability of the program. **Comments** - write the purpose and your basic idea of each function at the beginning of each function, and if you declare variables by yourself, write comments to explain the meaning of the variable.

2. **Test cases**: For each problem, give the results of the testing of your programs using the test cases specified in this manual. You may also use other test cases to show the testing of your programs.

3. **Discussion**: For each problem, give an analysis and discussion on the problem and the possible solutions, and any further considerations when implementing the functions and programs. You may write the discussion based on your understanding on the problem,

logical analysis of the problem and your solution. This section aims to see how much you understand the problems and your suggested solutions.

Submission: submit your report to the **Software Projects Lab (N4-B1B-11)** before the deadline.

IV. Source Code and Report (softcopy) Submission

1. Please put the source programs (**p1.c**, **p2.c**, **p3.c**, and **p4.c** only) into a zip file with file name using your group number and your name (e.g. **FS3_Bill Gates**). Please make sure that the source programs for the problems are correctly named according to this manual
2. Please put the report with file name using your group number and your name (e.g. **FS3_Bill Gates**).
3. Upload the source file and report (softcopy) into the correct folder (according to your lab group) through NTULearn. If you are not sure how to do this, please go to **Software Projects Lab (N4-B1B-11)** and check with the lab supervisor or technician.
4. Upload the source codes to the online **PSA** submission system (Please check user manual for more details [**Using PSA System – Assignment submission.pdf]**)
5. Please make sure that you upload your programs before the deadline.

V. Marking Scheme

The marking of each problem of the assignment will be based on the following criteria:

1. Correctness of the program.
2. Presentation and documentation of the C codes (whether the program is properly indented, and whether the program is well commented to aid understanding).
3. Discussion and analysis on the problem and solutions, and any further considerations when implementing the functions and programs.

VI. Plagiarism

Please be reminded that PLAGIARISM (or copying part of/complete assignment) is considered as CHEATING, which is strictly prohibited. You will get zero mark on your assignment if you are found guilty of plagiarism.

C. Appendix -- PROGRAM TEMPLATE

```
/* CX1007 Data Structures
   2015/16 S1
   Author and Lab Group: yourname labgroup
   Program name: labgroup_yourname.c
   Date: xx November 2015
   Purpose: Implementing the required functions for Assignment 1*/
```

Question No 1.

```
#include <stdio.h>
#include <stdlib.h>


////////////////////////////////////////////////////////////////////////////////

typedef struct _listnode{
        int item;
        struct  listnode *next;
} ListNode;                    // You should not change the definition of ListNode


typedef struct _linkedlist{
        int size;
        ListNode *head;
} LinkedList;                  // You should not change the definition of LinkedList


////////////////////////// function prototypes /////////////////////////////////////

// This is for question 1. You should not change the prototype of this function
int insertSortedLinkedList(LinkedList *ll, int item);

// You may use the following functions or you may write your own
void printList(LinkedList *ll);
void removeAllItems(LinkedList *ll);
ListNode * findNode(LinkedList *ll, int index);
int insertNode(LinkedList *ll, int index, int value);
int removeNode(LinkedList *ll, int index);


///////////////////////////// main() //////////////////////////////////////////////

int main()
{
        LinkedList ll;
        int c, i, j;

        j = -1;

        //Initialize the linked list as an empty linked list
        ll.head = NULL;
        ll.size = 0;

        printf("1: Insert an integer to the sorted linked list:\n");
        printf("2: Print the index of most recent input value:\n");
        printf("3: Print sorted linked list:\n");
        printf("0: Quit:\n");
        printf("Please input your choice(1/2/3/0): ");
        scanf("%d", &c);

        while (c != 0)
        {
                switch (c)
                {
                case 1:
                        printf("Input an integer that you want to add to the linked list:
");
                        scanf("%d", &i);
                        j = insertSortedLinkedList(&ll, i); // You need to code this
function
                        printf("The resulting linked list is: ");
                        printList(&ll);
                        break;
                case 2:
                        if (j > -1)
```

```
                                printf("The value %d was added at index %d\n", i, j);
                        else
                                printf("Nothing to print! \n");
                        break;
                case 3:
                        printf("The resulting linked list is: ");
                        printList(&ll);
                        break;
                case 0:
                        removeAllItems(&ll);
                        break;
                default:
                        printf("Choice unknown;\n");
                        break;
                }

                printf("\nPlease input your choice(1/2/3/0): ");
                scanf("%d", &c);
        }

        return 0;
}


////////////////////////////////////////////////////////////////////////////////

int insertSortedLinkedList(LinkedList *ll, int item)
{
        /* add your code here */
}

void printList(LinkedList *ll)
{
        ListNode *cur;
        if (ll == NULL)
                return;
        cur = ll->head;
        if (cur == NULL)
                printf("Empty");
        while (cur != NULL)
        {
                printf("%d ", cur->item);
                cur = cur->next;
        }
        printf("\n");
}

void removeAllItems(LinkedList *ll)
{
        ListNode *cur = ll->head;
        ListNode *tmp;

        while (cur != NULL){
                tmp = cur->next;
                free(cur);
                cur = tmp;
        }
        ll->head = NULL;
        ll->size = 0;
}

ListNode * findNode(LinkedList *ll, int index)
{
        ListNode *temp;

        if (ll == NULL || index < 0 || index >= ll->size)
                return NULL;

        temp = ll->head;

        if (temp == NULL || index < 0)
                return NULL;

        while (index > 0){
                temp = temp->next;
                if (temp == NULL)
                        return NULL;
                index--;
        }
```

```
        return temp;
}

int insertNode(LinkedList *ll, int index, int value)
{
        ListNode *pre, *cur;

        if (ll == NULL || index < 0 || index > ll->size + 1)
                return -1;

        // If empty list or inserting first node, need to update head pointer
        if (ll->head == NULL || index == 0){
                cur = ll->head;
                ll->head = malloc(sizeof(ListNode));
                ll->head->item = value;
                ll->head->next = cur;
                ll->size++;
                return 0;
        }

        // Find the nodes before and at the target position
        // Create a new node and reconnect the links
        if ((pre = findNode(ll, index - 1)) != NULL){
                cur = pre->next;
                pre->next = malloc(sizeof(ListNode));
                pre->next->item = value;
                pre->next->next = cur;
                ll->size++;
                return 0;
        }

        return -1;
}

int removeNode(LinkedList *ll, int index)
{
        ListNode *pre, *cur;

        // Highest index we can remove is size-1
        if (ll == NULL || index < 0 || index >= ll->size)
                return -1;

        // If removing first node, need to update head pointer
        if (index == 0){
                cur = ll->head->next;
                free(ll->head);
                ll->head = cur;
                ll->size--;

                return 0;
        }

        // Find the nodes before and after the target position
        // Free the target node and reconnect the links
        if ((pre = findNode(ll, index - 1)) != NULL){

                if (pre->next == NULL)
                        return -1;

                cur = pre->next;
                pre->next = cur->next;
                free(cur);
                ll->size--;
                return 0;
        }

        return -1;
}
```

Question No 2.

```
#include <stdio.h>
#include <stdlib.h>


////////////////////////////////////////////////////////////////////////////////

typedef struct _listnode
{
	int item;
	struct _listnode *next;
} ListNode;                    // You should not change the definition of ListNode

typedef struct _linkedlist
{
	int size;
	ListNode *head;
} LinkedList;                  // You should not change the definition of LinkedList


////////////////////////// function prototypes /////////////////////////////////////

// These are for question 2. You should not change the prototypes of these functions
void sortedMergeLinkedList(LinkedList *ll1, LinkedList *ll2, LinkedList
*resultMergedList);
int insertSortedLinkedList(LinkedList *ll, int item);

// You may use the following functions or you may write your own
void printList(LinkedList *ll);
void removeAllItems(LinkedList *ll);
ListNode * findNode(LinkedList *ll, int index);
int insertNode(LinkedList *ll, int index, int value);
int removeNode(LinkedList *ll, int index);


///////////////////////////// main() //////////////////////////////////////////////////

int main()
{
	LinkedList ll1, ll2, resultMergedList;
	int c, i, j;

	//Initialize the linked list 1 as an empty linked list
	ll1.head = NULL;
	ll1.size = 0;

	//Initialize the linked list 2 as an empty linked list
	ll2.head = NULL;
	ll2.size = 0;

	//Initialize the merged linked list as an empty linked list
	resultMergedList.head = NULL;
	resultMergedList.size = 0;

	printf("1: Insert an integer to the sorted linked list 1:\n");
	printf("2: Insert an integer to the sorted linked list 2:\n");
	printf("3: Print sorted merged linked list:\n");
	printf("0: Quit:\n");
	printf("Please input your choice(1/2/3/0): ");
	scanf("%d", &c);

	while (c != 0)
	{
		switch (c)
		{
		case 1:
			printf("Input an integer that you want to add to the linked list 1:
");
			scanf("%d", &i);
			j = insertSortedLinkedList(&ll1, i); // You need to code this
function
			printf("Linked list 1: ");
			printList(&ll1);
			break;
		case 2:
			printf("Input an integer that you want to add to the linked list 2:
");
```

```
                        scanf("%d", &i);

                        j = insertSortedLinkedList(&ll2, i); // You need to code this
function
                        printf("Linked list 2: ");
                        printList(&ll2);
                        break;
                case 3:
                        sortedMergeLinkedList(&ll1, &ll2, &resultMergedList); // You need
to code this function
                        printf("The resulting linked list is: ");
                        printList(&resultMergedList);
                        break;
                case 0:
                        removeAllItems(&ll1);
                        removeAllItems(&ll2);
                        removeAllItems(&resultMergedList);
                        break;
                default:
                        printf("Choice unknown;\n");
                        break;
                }

                printf("\nPlease input your choice(1/2/3/0): ");
                scanf("%d", &c);
        }
        return 0;
}


////////////////////////////////////////////////////////////////////////////////
```

```
void sortedMergeLinkedList(LinkedList *ll1, LinkedList *ll2, LinkedList
*resultMergedList)
{
        /* add your code here */
}

int insertSortedLinkedList(LinkedList *ll, int item)
{
     /* add your code here */
}
```

```
void printList(LinkedList *ll)
{
        ListNode *cur;
        if (ll == NULL)
                return;
        cur = ll->head;
        if (cur == NULL)
                printf("Empty");
        while (cur != NULL)
        {
                printf("%d ", cur->item);
                cur = cur->next;
        }
        printf("\n");
}

void removeAllItems(LinkedList *ll)
{
        ListNode *cur = ll->head;
        ListNode *tmp;

        while (cur != NULL){
                tmp = cur->next;
                free(cur);
                cur = tmp;
        }
        ll->head = NULL;
        ll->size = 0;
}

ListNode * findNode(LinkedList *ll, int index)
{
        ListNode *temp;

        if (ll == NULL || index < 0 || index >= ll->size)
                return NULL;
```

```
	temp = ll->head;

	if (temp == NULL || index < 0)
		return NULL;

	while (index > 0){
		temp = temp->next;
		if (temp == NULL)
			return NULL;
		index--;
	}

	return temp;
}

int insertNode(LinkedList *ll, int index, int value)
{
	ListNode *pre, *cur;

	if (ll == NULL || index < 0 || index > ll->size + 1)
		return -1;

	// If empty list or inserting first node, need to update head pointer
	if (ll->head == NULL || index == 0){
		cur = ll->head;
		ll->head = malloc(sizeof(ListNode));
		ll->head->item = value;
		ll->head->next = cur;
		ll->size++;
		return 0;
	}

	// Find the nodes before and at the target position
	// Create a new node and reconnect the links
	if ((pre = findNode(ll, index - 1)) != NULL){
		cur = pre->next;
		pre->next = malloc(sizeof(ListNode));
		pre->next->item = value;
		pre->next->next = cur;
		ll->size++;
		return 0;
	}

	return -1;
}

int removeNode(LinkedList *ll, int index)
{
	ListNode *pre, *cur;

	// Highest index we can remove is size-1
	if (ll == NULL || index < 0 || index >= ll->size)
		return -1;

	// If removing first node, need to update head pointer
	if (index == 0){
		cur = ll->head->next;
		free(ll->head);
		ll->head = cur;
		ll->size--;
		return 0;
	}

	// Find the nodes before and after the target position
	// Free the target node and reconnect the links
	if ((pre = findNode(ll, index - 1)) != NULL){

		if (pre->next == NULL)
			return -1;

		cur = pre->next;
		pre->next = cur->next;
		free(cur);
		ll->size--;
		return 0;
	}
	return -1;
}
```

Question No 3.

```
#include <stdio.h>
#include <stdlib.h>


//////////////////////////////////////////////////////////////////////////////

typedef struct _listnode
{
        int item;
        struct _listnode *next;
} ListNode;     // You should not change the definition of ListNode

typedef struct _linkedlist
{
        int size;
        ListNode *head;
} LinkedList;  // You should not change the definition of LinkedList

typedef struct _queue
{
        LinkedList ll;
} Queue;        // You should not change the definition of Queue


////////////////////////// function prototypes /////////////////////////////////////

// These are for question 3. You should not change the prototype of these functions
void createQueueFromLinkedList(LinkedList *ll, Queue * q);
void removeEvenValues(Queue *q);
void enqueue(Queue *q, int item);
int dequeue(Queue *q);
int isEmptyQueue(Queue *s);

// You may use the following functions or you may write your own
void printList(LinkedList *ll);
void removeAllItems(LinkedList *ll);
ListNode * findNode(LinkedList *ll, int index);
int insertNode(LinkedList *ll, int index, int value);
int removeNode(LinkedList *ll, int index);
void removeAllItemsFromQueue(Queue *q);


///////////////////////////// main() //////////////////////////////////////////////////

int main()
{
        int c, i, j;
        LinkedList ll;
        Queue q;

        j = -1;

        // Initialize the linked list as an empty linked list
        ll.head = NULL;
        ll.size = 0;

        // Initialize the queue as an empty queue
        q.ll.head = NULL;
        q.ll.size = 0;

        printf("1: Insert an integer into the linked list:\n");
        printf("2: Create queue from the linked list:\n");
        printf("3: Remove even numbers from queue:\n");
        printf("0: Quit:\n");
        printf("Please input your choice(1/2/3/0): ");
        scanf("%d", &c);

        while (c != 0)
        {
                switch (c)
                {
                case 1:
                        printf("Input an integer that you want to add to the linked list:
");
                        scanf("%d", &i);
                        insertNode(&ll, ll.size, i);
                        printf("The resulting linked list is: ");
```

```
                        printList(&ll);
                        break;
                case 2:
                        createQueueFromLinkedList(&ll, &q); // You need to code this
function
                        printf("The resulting queue is: ");
                        printList(&q.ll);
                        break;
                case 3:
                        removeEvenValues(&q); // You need to code this function
                        printf("The resulting queue after removing even integers is: ");
                        printList(&q.ll);
                        break;
                case 0:
                        removeAllItems(&ll);
                        removeAllItemsFromQueue(&q);
                        break;
                default:
                        printf("Choice unknown;\n");
                        break;
                }

                printf("\nPlease input your choice(1/2/3/0): ");
                scanf("%d", &c);
        }

        return 0;
}


////////////////////////////////////////////////////////////////////////////////
```

```
void createQueueFromLinkedList(LinkedList *ll, Queue * q)
{
        /* add your code here */
}

void removeEvenValues(Queue *q)
{
        /* add your code here */
}

void enqueue(Queue *q, int item)
{
        /* add your code here */
}

int dequeue(Queue *q)
{
        /* add your code here */
}

int isEmptyQueue(Queue *q)
{
        /* add your code here */
}
```

```
void removeAllItemsFromQueue(Queue *q)
{
        int count, i;
        if (q == NULL)
                return;
        count = q->ll.size;

        for (i = 0; i < count; i++)
                dequeue(q);
}

void printList(LinkedList *ll)
{
        ListNode *cur;
        if (ll == NULL)
                return;
        cur = ll->head;
        if (cur == NULL)
                printf("Empty");
        while (cur != NULL)
        {
                printf("%d ", cur->item);
```

```
                cur = cur->next;
        }
        printf("\n");
}

void removeAllItems(LinkedList *ll)
{
        ListNode *cur = ll->head;
        ListNode *tmp;

        while (cur != NULL){
                tmp = cur->next;
                free(cur);
                cur = tmp;
        }
        ll->head = NULL;
        ll->size = 0;
}

ListNode * findNode(LinkedList *ll, int index)
{
        ListNode *temp;

        if (ll == NULL || index < 0 || index >= ll->size)
                return NULL;

        temp = ll->head;

        if (temp == NULL || index < 0)
                return NULL;

        while (index > 0){
                temp = temp->next;
                if (temp == NULL)
                        return NULL;
                index--;
        }

        return temp;
}

int insertNode(LinkedList *ll, int index, int value)
{
        ListNode *pre, *cur;

        if (ll == NULL || index < 0 || index > ll->size + 1)
                return -1;

        // If empty list or inserting first node, need to update head pointer
        if (ll->head == NULL || index == 0){
                cur = ll->head;
                ll->head = malloc(sizeof(ListNode));
                ll->head->item = value;
                ll->head->next = cur;
                ll->size++;
                return 0;
        }

        // Find the nodes before and at the target position
        // Create a new node and reconnect the links
        if ((pre = findNode(ll, index - 1)) != NULL){
                cur = pre->next;
                pre->next = malloc(sizeof(ListNode));
                pre->next->item = value;
                pre->next->next = cur;
                ll->size++;
                return 0;
        }

        return -1;
}

int removeNode(LinkedList *ll, int index)
{
        ListNode *pre, *cur;

        // Highest index we can remove is size-1
        if (ll == NULL || index < 0 || index >= ll->size)
                return -1;
```

```
        // If removing first node, need to update head pointer
        if (index == 0){

                cur = ll->head->next;
                free(ll->head);
                ll->head = cur;
                ll->size--;

                return 0;
        }

        // Find the nodes before and after the target position
        // Free the target node and reconnect the links
        if ((pre = findNode(ll, index - 1)) != NULL){

                if (pre->next == NULL)
                        return -1;

                cur = pre->next;
                pre->next = cur->next;
                free(cur);
                ll->size--;
                return 0;
        }

        return -1;
}
```

Question No 4.

```
#include <stdio.h>
#include <stdlib.h>


//////////////////////////////////////////////////////////////////////////////////

typedef struct  listnode{
        int item;
        struct _listnode *next;
} ListNode;                    // You should not change the definition of ListNode

typedef struct  linkedlist{
        int size;
        ListNode *head;
} LinkedList;                  // You should not change the definition of LinkedList


//////////////////////// function prototypes /////////////////////////////////////

// This is for question 4. You should not change the prototype of this function
void frontBackSplitLinkedList(LinkedList* ll, LinkedList *resultFrontList, LinkedList
*resultBackList);

// You may use the following functions or you may write your own
void printList(LinkedList *ll);
void removeAllItems(LinkedList *l);
ListNode * findNode(LinkedList *ll, int index);
int insertNode(LinkedList *ll, int index, int value);
int removeNode(LinkedList *ll, int index);

///////////////////////////// main() //////////////////////////////////////////////

int main()
{
        int c, i;
        LinkedList ll;
        LinkedList resultFrontList, resultBackList;

        //Initialize the linked list as an empty linked list
        ll.head = NULL;
        ll.size = 0;

        //Initialize the front linked list as an empty linked list
        resultFrontList.head = NULL;
        resultFrontList.size = 0;
```

```
        // Initialize the back linked list as an empty linked list
        resultBackList.head = NULL;
        resultBackList.size = 0;

        printf("1: Insert an integer to the linked list:\n");
        printf("2: Print the linked list:\n");
        printf("3: Split the linked list into two linked lists, frontList and
backList:\n");
        printf("0: Quit:\n");
        printf("Please input your choice(1/2/3/0): ");
        scanf("%d", &c);

        while (c != 0)
        {
                switch (c)
                {
                case 1:
                        printf("Input an integer that you want to add to the linked list:
");
                        scanf("%d", &i);
                        insertNode(&ll, ll.size, i);
                        printf("The resulting linked list is: ");
                        printList(&ll);
                        break;
                case 2:
                        printf("The resulting linked list is: ");
                        printList(&ll);
                        break;
                case 3:


                        printf("The resulting linked lists after splitting the given linked
list are:\n");
                        frontBackSplitLinkedList(&ll, &resultFrontList, &resultBackList); //
You need to code this function
                        printf("Front linked list: ");
                        printList(&resultFrontList);
                        printf("Back linked list: ");
                        printList(&resultBackList);
                        printf("\n");
                        break;
                case 0:
                        removeAllItems(&ll);
                        removeAllItems(&resultFrontList);
                        removeAllItems(&resultBackList);
                        break;
                default:
                        printf("Choice unknown;\n");
                        break;
                }

                printf("\nPlease input your choice(1/2/3/0): ");
                scanf("%d", &c);
        }

        return 0;
}


////////////////////////////////////////////////////////////////////////////////////////

 void frontBackSplitLinkedList(LinkedList* ll, LinkedList * resultFrontList,
 LinkedList *resultBackList)
 {
        /* add your code here */
 }

void printList(LinkedList *ll)
{
        ListNode *cur;
        if (ll == NULL)
                return;
        cur = ll->head;
        if (cur == NULL)
                printf("Empty");
        while (cur != NULL)
        {
                printf("%d ", cur->item);
                cur = cur->next;
```

```
        }
        printf("\n");
}

void removeAllItems(LinkedList *ll)
{
        ListNode *cur = ll->head;
        ListNode *tmp;

        while (cur != NULL){
                tmp = cur->next;
                free(cur);
                cur = tmp;
        }
        ll->head = NULL;
        ll->size = 0;
}

ListNode * findNode(LinkedList *ll, int index)
{
        ListNode *temp;

        if (ll == NULL || index < 0 || index >= ll->size)
                return NULL;

        temp = ll->head;

        if (temp == NULL || index < 0)
                return NULL;

        while (index > 0){


                temp = temp->next;
                if (temp == NULL)
                        return NULL;
                index--;
        }

        return temp;
}

int insertNode(LinkedList *ll, int index, int value)
{
        ListNode *pre, *cur;

        if (ll == NULL || index < 0 || index > ll->size + 1)
                return -1;

        // If empty list or inserting first node, need to update head pointer
        if (ll->head == NULL || index == 0){
                cur = ll->head;
                ll->head = malloc(sizeof(ListNode));
                ll->head->item = value;
                ll->head->next = cur;
                ll->size++;
                return 0;
        }

        // Find the nodes before and at the target position
        // Create a new node and reconnect the links
        if ((pre = findNode(ll, index - 1)) != NULL){
                cur = pre->next;
                pre->next = malloc(sizeof(ListNode));
                pre->next->item = value;
                pre->next->next = cur;
                ll->size++;
                return 0;
        }

        return -1;
}

int removeNode(LinkedList *ll, int index)
{
        ListNode *pre, *cur;

        // Highest index we can remove is size-1
        if (ll == NULL || index < 0 || index >= ll->size)
                return -1;
```

```
        // If removing first node, need to update head pointer
        if (index == 0){
                cur = ll->head->next;
                free(ll->head);
                ll->head = cur;
                ll->size--;

                return 0;
        }

        // Find the nodes before and after the target position
        // Free the target node and reconnect the links
        if ((pre = findNode(ll, index - 1)) != NULL){

                if (pre->next == NULL)
                        return -1;

                cur = pre->next;
                pre->next = cur->next;
                free(cur);
                ll->size--;
                return 0;
        }

        return -1;
}
```