Problems on Linked Lists:

<u>Section A:</u>

1. Write a recursive routine to count the number of nodes in a linked list.

2. Write a routine which returns 1 if any node of the linked list LL contains the data value 650, otherwise it returns 0.

3. Write a function to print the value stored in the last element of a linked list pHead.

4. Given a list pFirst, write a function to add a node with value 800 at the end of the list.

5. Write a function to delete the first node of the linked list PP.

6. The nodes of a linked list LL, have integer values in ascending order. Write a function to delete the node with value 90 from the list LL.

<u>Section B:</u>

1- Write a Count() function that counts the number of times a given integer occurs in a list.

```
int Count(struct node* head, int searchFor)
```

2- Write a function GetNth( ) that takes a linked list and an integer index and returns the data value stored in the node at that index position. GetNth() uses the C numbering convention that the first node is index 0, the second is index 1, ... and so on. So for the list {42, 13, 666} GetNth() with index 1 should return 13. The index should be in the range [0..length-1]. If it is not, GetNth() should print an error.

```
int GetNth(struct node* head, int index)
```

3- Write a function DeleteList() that takes a list, deallocates all of its memory and sets its head pointer to NULL (the empty list).

```
void DeleteList(struct node** headRef)
```

4- Write a Pop() function that takes a list, deletes the head node, and returns the head node's data. If the list is empty, it prints out "LIST IS EMPTY".

```
int Pop(struct node** headRef)
```

5- Write a function InsertNth( ) which inserts a new node at a specified index within a list. The caller may specify any index in the range [0..length], and the new node should be inserted so as to be at that index position.

```
void InsertNth(struct node** P, int index, int data)
```

6- Write an Append() function that takes two lists, 'a' and 'b', appends 'b' onto the end of 'a', and then sets 'b' to NULL (since it is now trailing off the end of 'a').

```
void Append(struct node** aRef, struct node** bRef)
```

7- Write a function Duplicates( ) that takes a list containing integers, and returns the list after removing all duplicate entries from the list. Thus given the list 20, 25, 25, 30, 40, 45, 45, 45, 60, it should return 20, 25, 30, 40, 45, 60.

```
struct node * Duplicates(struct node* H )
```

<u>Section C:</u>

1. Given a linked list pointed by *A, write a function trim which deletes the first node. In all linked list problems, always take care of the situation that the list may have only one node or that the list may be empty.

```
struct node *trim(struct node *A)
```

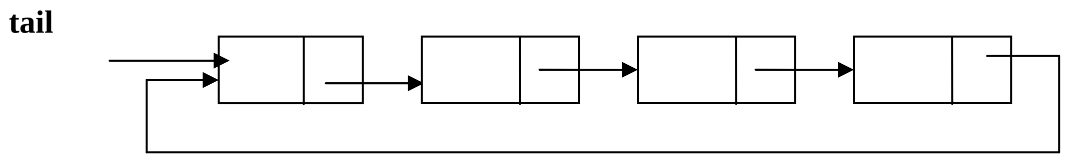
2. Repeat problem 1 by writing the following function:

```
 void trim(struct node * *A)
```

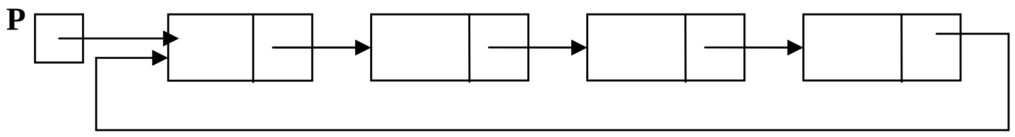
3. Given a linked list *B, write a function to add 10 to the last element.

```
struct node *add_last(struct node *B)
```

4. A circular list is being pointed to by its tail. Count the number of nodes in the list.



5. Consider the following *circular list,* where p points to the FIRST node of the list.



Complete the following incomplete function to extend the circular linked list being pointed by p (the first node) by appending a singly linked list (pointed by q) to the end of the circular list. The resulting list will be one big circular list.

It is specified that the Circular list is NOT empty i.e. p cannot be NULL, but q may be NULL.

- Desired: the first node of the original circular list will be the first node of the newly created big circular list.

```
void extend(struct node *p, struct node * q)
{
   struct node *t; /* t is used to traverse the lists */
   if (    _____________ )  /* if standard linked list is
                                     empty */
      return;

   t = p;      /* Assume p cannot be NULL */


  while ( __________  )  /*traverse the circular list */
      t =______________;


  ____________; /* append the standard list */
   while (______________) /* traverse the std. list */


      t = _______________;
     _____________________;
```

6. Reverse a given list, making use of a stack. Use the stack functions done in the class. (Stack code given on the webpage.)

Let A be the given list. Put the nodes of the stack in stackA. Then pop them up one by one and append to another list B.

7. Write a function which accepts two linearly linked lists Alpha and Beta, moves the first node of Beta to front of Alpha and returns the lists back to the main program.