LAB - 7 : (Doubly Linked List) (11/2/24)

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

struct Node
{
  int data;
  struct Node *prev;
  struct Node *next;
};

struct Node *createNode(int data)
{
  struct Node * newNode = (struct Node*) malloc (sizeof(struct Node));
  if (newNode == NULL)
  {
      printf("Memory allocation failed\n");
      exit(1);
  }
  newNode -> data = data;
  newNode -> prev = NULL;
  newNode -> next = NULL;
  return newNode;
}

void insertNodetoLeft (struct Node *head, struct Node* forget,
                       int data)

{
  struct Node * newNode = createNode(data);
  if (forget -> prev != NULL)
  {   forget -> prev -> next = newNode;
      newNode -> prev = forget -> prev;
  }
```

```
        else
        {
            head = newNode;
        }
        newNode->next = target;
        target->prev = newNode;
    }

void deleteNode(struct Node *head, int value)
{
    struct Node *current = head;
    while (current != NULL)
    {
        if(current->data == value)
        {
            if (current->prev != NULL)
            {
                current->prev->next = current->next;
            }
            else
            {
                head = current->next;
            }
            if (current->next != NULL)
            {
                current->next->prev = current->prev;
            }
            free(current);
            return;
        }
        current = current->next;
    }
    printf("Node with value %d not found\n", value);
}
```

```
void displayList(struct Node* head)
{
  printf("Doubly Linked List: ");
  while(head!=NULL)
  {
     printf("%d <->", head->data);
     head = head->next;
  }
  printf("NULL\n");
}

int main()
{
   struct Node *head = NULL;
   head = createNode(1);
   hedd->next = createNode(2);
   head->next->prev = head;
   head->next->next = createNode(3);
   head->next->next->prev = head->next;

   displayList(head);
   insertNodeToLeft(head, head->next, 10);
   printf("After insertion:\n");
   displayList(head);

   deleteNode(head, 2);
   printf("After deletion:\n");
   displayList(head);

   return 0;

}
```

Output:

Doubly Linked List : 1 ↔ 2 ↔ 3 ↔ NULL
After insertion:
Doubly Linked List : 1 ↔ 10 ↔ 2 ↔ 3 ↔ NULL
After deletion:
Doubly Linked List : 1 ↔ 10 ↔ 3 ↔ NULL

Leetcode Problem (Split Linked List in Parts) (1/2/24)

```
int getLength(struct ListNode* head)
{
  int length = 0;
  while (head != NULL)
  {
    length++;
    head = head->next;
  }
  return length;
}

struct ListNode** splitListToParts(struct ListNode* head,
                                   int k, int *returnSize)
{
  int length = getLength(head);
  int partsize = length / k;
  int remainder = length % k;
  struct ListNode** result = (struct ListNode**) malloc
                             (k * sizeof(struct ListNode*));
  *returnSize = k;
```

1/2/2024

```
for (int i = 0 ; i < k ; i++)
{
    int currentPartSize = partSize + (i < remainder ?
                                      1 : 0);
    if (currentPartSize == 0)
    {
        result[i] = NULL;
    }
    else
    {
        result[i] = head;
        for (int j = 0 ; j < currentPartSize - 1 ; j++)
        {
            head = head -> next;
        }
        struct ListNode* temp = head -> next;
        head -> next = NULL;
        head = temp;
    }
}
return result;
}
```