7a) WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.

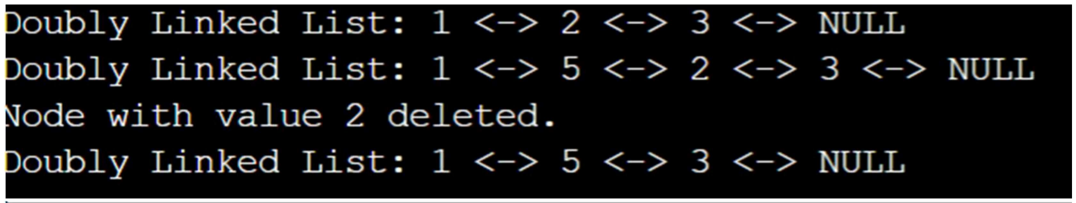
b) Insert a new node to the left of the node.

c) Delete the node based on a specific value

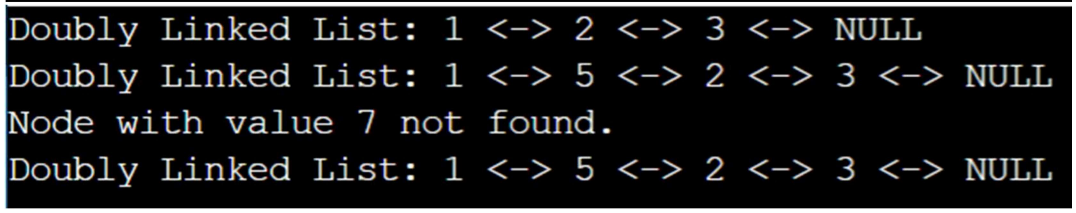
Display the contents of the list.

```
#include <stdio.h>

#include <stdlib.h>

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


struct Node* createNode(int data) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->prev = newNode->next = NULL; return newNode;
}


void insertLeft(struct Node** head, struct Node* target, int data) { struct
  Node* newNode = createNode(data);
  newNode->next = target ? target : *head; if (target) {
    newNode->prev = target->prev; target->prev = newNode;
  }
  if (newNode->prev)
    newNode->prev->next = newNode; else
    *head = newNode;
```

```
}

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

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

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


    insertLeft(&head, head->next, 5); displayList(head);


    deleteNode(&head, 2); displayList(head);


    return 0;

}
```



```
Doubly Linked List: 1 <-> 2 <-> 3 <-> NULL
Doubly Linked List: 1 <-> 5 <-> 2 <-> 3 <-> NULL
Node with value 2 deleted.
Doubly Linked List: 1 <-> 5 <-> 3 <-> NULL
```



```
Doubly Linked List: 1 <-> 2 <-> 3 <-> NULL
Doubly Linked List: 1 <-> 5 <-> 2 <-> 3 <-> NULL
Node with value 7 not found.
Doubly Linked List: 1 <-> 5 <-> 2 <-> 3 <-> NULL
```

Q7b) Given pointers to the heads of two sorted linked lists, merge them into a single, sorted linked list. Either head pointer may be null meaning that the corresponding list is empty.

```
SinglyLinkedListNode* mergeLists(SinglyLinkedListNode* head1,
  SinglyLinkedListNode* head2) { SinglyLinkedListNode* mergedHead =
(SinglyLinkedListNode*)malloc(sizeof(SinglyLinkedListNode));
  SinglyLinkedListNode* tail = mergedHead;
  mergedHead->next = NULL;

  while (head1 != NULL && head2 != NULL) { if (head1->data <=
    head2->data) {
      tail->next = head1; head1 = head1->next;
    } else {
      tail->next = head2; head2 = head2->next;
    }
    tail = tail->next;
  }

  // Attach the remaining nodes of the non-empty list tail->next = (head1
  != NULL) ? head1 : head2;

  // Save and remove the dummy node SinglyLinkedListNode* result =
  mergedHead->next; free(mergedHead);

  return result;
```

}

Sample Input

```
1
3
1
2
3
2
3
4
```

Sample Output

```
1 2 3 3 4
```

Explanation

The first linked list is: $1 \rightarrow 3 \rightarrow 7 \rightarrow NULL$

The second linked list is: $3 \rightarrow 4 \rightarrow NULL$

Hence, the merged linked list is: $1 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 4 \rightarrow NULL$