LAB 4 - (2)Qns - Dividing List into 2 parts - Create CLL : Sort, Order

divide the list into two parts

Create a Doubly Linked List, after creation of the list divide the list into two Circular Doubly Linke List from the given node value Keep the order of first list the same and also include the given node in first list. And the second list should be in reverse order.

To achieve this, you need to implement a function “append” list and them implement the function “divideAt”

Note: append function should insert node at end of the list.

Utilize the provided boilerplate code or develop your own, ensuring it aligns with the given input and output formats.

Minimize extraneous print statements; your output should match the specified format.

Input Format

First Line consist of a number N showing the number of Nodes Second Line consist of the N space separated integers The third line consist of k: An integer representing the number where the list will be divided

Constraints

N<100 k is a number that is present in the list the nodes are integers

Output Format

The program will print the first part of list in First line and the second part (reversed part) in next line.

Sample Input 0

```
5
5 1 3 2 4
3
```

Sample Output 0

```
5 1 3
4 2
```

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    if (newNode != NULL) {
        newNode->data = data;
        newNode->next = NULL;
        newNode->prev = NULL;
    }
    return newNode;
}

void append(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return;
    }

    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}

void divideAt(struct Node* originalHead, struct Node** firstHead, struct Node** secondHead, int nodeData) {
    struct Node* current = originalHead;
    while (current != NULL) {
        if (current->data == nodeData) {
            *firstHead = originalHead;
            *secondHead = current->next;

            if (*secondHead != NULL) {
                (*secondHead)->prev = NULL;
            }
            current->next = NULL;
            return;
        }
        current = current->next;
    }
}

void display(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
}

void reverseAndDisplay(struct Node* head) {
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }

    while (current != NULL) {
        printf("%d ", current->data);
        current = current->prev;
    }
}

int main() {
    struct Node* dll = NULL;

    int numElements;
```

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

    for (int i = 0; i < numElements; i++) {
        int data;
        scanf("%d", &data);
        append(&dll, data);
    }

    int nodeDataToDivideAt;
    scanf("%d", &nodeDataToDivideAt);

    struct Node* firstList = NULL;
    struct Node* secondList = NULL;

    divideAt(dll, &firstList, &secondList, nodeDataToDivideAt);

    display(firstList);
    printf("\n");

    if (secondList) {
        reverseAndDisplay(secondList);
        printf("\n");
    } else {
        printf("NULL\n");
    }
    while (dll != NULL) {
        struct Node* temp = dll;
        dll = dll->next;
        free(temp);
    }

    return 0;
}
```

---

create_cll_sorted_order

Create a Circular Linked List (CLL) in ascending order. Given N number of nodes create the CLL in ascending order. To achieve this, you need to implement a function “insertSorted”.

Note: Even Duplicate node values are also present consider them while writing the logic.

Utilize the provided boilerplate code or develop your own, ensuring it aligns with the given input and output formats.

Minimize extraneous print statements; your output should match the specified format.

Input Format

First Line consist of a number N showing the number of Nodes

Second Line consist of the N space separated integers

Constraints

N<100

nodes can be any interger.

Output Format

N space seprated intergers.

Sample Input 0

```
5
2 1 3 4 5
```

Sample Output 0

```
1 2 3 4 5
```

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

// Define a structure for a node in the CLL
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node into the CLL in ascending order
struct Node* insertSorted(struct Node* head, int value) {
    struct Node* newNode = createNode(value);
    if (head == NULL) {
        newNode->next = newNode; // Make it point to itself
        return newNode;
    }

    struct Node* current = head;
    struct Node* prev = NULL;

    do {
        if (current->data >= value) {
            newNode->next = current;
            if (prev != NULL)
                prev->next = newNode;
            else {
                // If we're inserting at the beginning
                struct Node* temp = head;
                while (temp->next != head)
                    temp = temp->next;
                temp->next = newNode;
            }
            if (current == head)
                return newNode;
            else
                return head;
        }
        prev = current;
        current = current->next;
    } while (current != head);

    // If the value is greater than all elements in the list
    prev->next = newNode;
    newNode->next = head;
    return head;
}

// Function to print the CLL
void printList(struct Node* head, int n) {
    struct Node* current = head;
    int count = 0;
    while (count < n) {
```

```
        printf("%d ", current->data);
        current = current->next;
        count++;
    }
    printf("\n");
}

int main() {
    int n;
    scanf("%d", &n);

    struct Node* head = NULL;

    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        head = insertSorted(head, value);
    }

    printList(head, n);

    return 0;
}
```