|  |
| --- |
| **NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**  **CS 201–DATA STRUCTURES LAB**  **Lab Session 07** |
| **Instructors:** Mr. Faizan Yousuf, Ms. Safia, Ms. Maham Mobin |

# Outline

* Circular Linked List
* Doubly Link List
* Exercise

**Doubly Linked List:**

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following is the important term to understand the concept of doubly linked list.

* Prev − Each link of a linked list contains a link to the previous link called Prev.

## **Doubly Linked List Representation**



## **Insertion Operation**

Following code demonstrates the insertion operation at the beginning of a doubly linked list.

### Example

Node addBefore(Node w, T x)

{

Node u = new Node();

u.x = x;

u.prev = w.prev;

u.next = w;

u.next.prev = u;

u.prev.next = u;

n++;

return u;

}

void add(int i, T x)

{

addBefore(getNode(i), x);

}

## **Deletion Operation**

Removing a node w from a DLList is easy. We only need to adjust pointers at w.next and w.prev so that they skip over w. Again, the use of the dummy node eliminates the need to consider any special cases:

### Example

void remove(Node w)

{

w.prev.next = w.next;

w.next.prev = w.prev;

n--;

}

Now the remove(i) operation is trivial. We find the node with index i and remove it:

### Example

T remove(int i)

{

Node w = getNode(i);

remove(w);

return w.x;

}

## **Insertion at the End of an Operation**

Following code demonstrates the insertion operation at the last position of a doubly linked list.

### Example

//insert link at the last location

void insertLast(int key,int data){

//create a link

struct node \*link =(struct node\*) malloc(sizeof(struct node));

link->key = key;

link->data = data;

if(isEmpty()){

//make it the last link

last= link;

}else{

//make link a new last link

last->next= link;

//mark old last node as prev of new link

link->prev =last;

}

//point last to new last node

last= link;

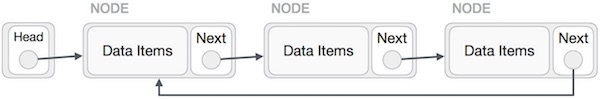
}

**Circular Linked List:**

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

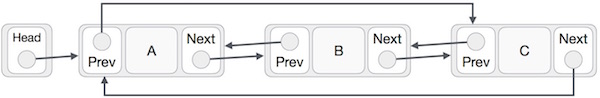
## **Singly Linked List as Circular**

In singly linked list, the next pointer of the last node points to the first node.



## **Doubly Linked List as Circular**

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.



## **Insertion Operation**

Following code demonstrates the insertion operation in a circular linked list based on single linked list.

### Example

//insert link at the first location

void insertFirst(int key,int data){

//create a link

struct node \*link =(struct node\*) malloc(sizeof(struct node));

link->key = key;

link->data= data;

if(isEmpty()){

head = link;

head->next= head;

}else{

//point it to old first node

link->next= head;

//point first to new first node

head = link;

}

}

## **Deletion Operation**

Following code demonstrates the deletion operation in a circular linked list based on single linked list.

//delete first item

struct node \* deleteFirst(){

//save reference to first link

struct node \*tempLink = head;

if(head->next== head){

head = NULL;

return tempLink;

}

//mark next to first link as first

head = head->next;

//return the deleted link

return tempLink;

}

## **Display List Operation**

Following code demonstrates the display list operation in a circular linked list.

//display the list

void printList(){

struct node \*ptr = head;

printf("\n[ ");

//start from the beginning

if(head != NULL){

while(ptr->next!= ptr){

printf("(%d,%d) ",ptr->key,ptr->data);

ptr = ptr->next;

}

}

printf(" ]");

}

**Exercise:**

**Question No. 1:**

Write a GetNth() function 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 assert() fail (or you could implement some other errorcase strategy).

void GetNthTest() {

struct node\* myList = BuildOneTwoThree(); // build {1, 2, 3}

int lastNode = GetNth(myList, 2); // returns the value 3

}

Essentially, GetNth() is similar to an array[i] operation — the client can ask for elements by index number. However, GetNth() no a list is much slower than [ ] on an array. The advantage of the linked list is its much more flexible memory management —we can Push() at any time to add more elements and the memory is allocated as needed.

// Given a list and an index, return the data

// in the nth node of the list. The nodes are numbered from 0.

// Assert fails if the index is invalid (outside 0..lengh-1).

int GetNth(struct node\* head, int index) {

// Your code

**Question No. 2:**

Your friend is an Intelligence officer at Pakistan Railway; his colleague gave him news about Karachi Express incident.

Incident: A group of People Hijack a Cabin of a Train, and one of their member is hidden somewhere in train.

Your task is to implement the scenario using double linked list to help your friend.

Step1: Find a Hijacked Cabin

Step2: Than Go back to the Engine and start finding the Last member

**Question No. 3:**

You are a Network Manager; your head asks you to implement a Series but Circular Networking between devices in admin office.

Your task is to implement the Scenario using Circular linked list.

The Network has 1 Router and 6 End Devices.