NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
CL-218–DATA STRUCTURES LAB

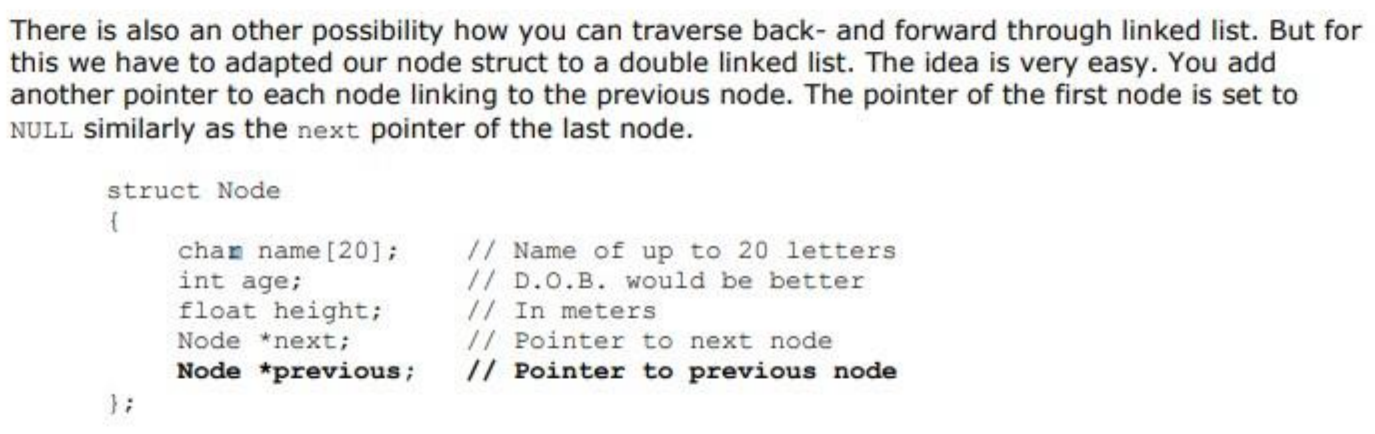
Lab Session 06

Instructors: MubashraFayyaz, Aqsa Zahid

Outline

- Circular Linked List
- Doubly Link List
- Exercise

IDEA:



There is also an other possibility how you can traverse back- and forward through linked list. But for this we have to adapted our node struct to a double linked list. The idea is very easy. You add another pointer to each node linking to the previous node. The pointer of the first node is set to `NULL` similarly as the `next` pointer of the last node.

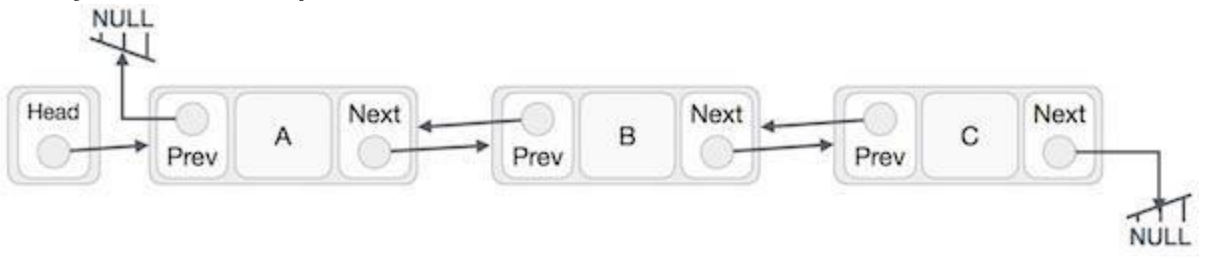
```
struct Node
{
     char name[20];      // Name of up to 20 letters
     int age;            // D.O.B. would be better
     float height;       // In meters
     Node *next;         // Pointer to next node
     Node *previous;     // Pointer to previous node
};
```

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(inti, 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(inti)
{
```

```
            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
voidinsertLast(intkey,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;
}
```

Following are advantages/disadvantages of doubly linked list over singly linked list.

Advantages over singly linked list

1) A DLL can be traversed in both forward and backward direction.
2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
3) We can quickly insert a new node before a given node.
In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

Disadvantages over singly linked list

1) Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though
2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in

following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

Insertion

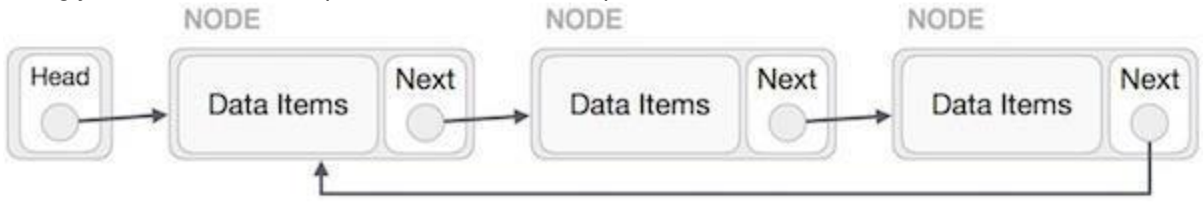
A node can be added in four ways
1) At the front of the DLL **2)** After a given node. **3)** At the end of the DLL **4)** Before a given node.

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.

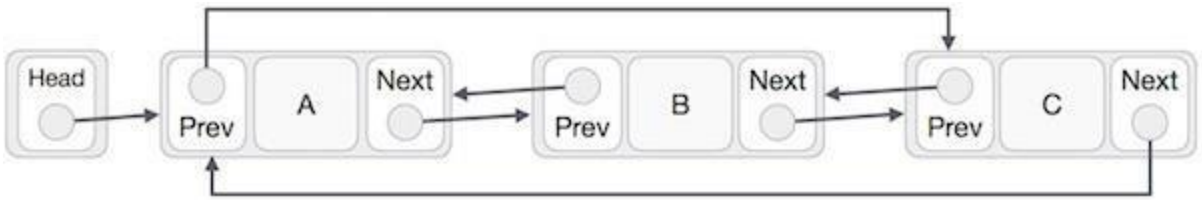
Singly Linked List as Circular

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 voidinsertFirst(intkey,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;
returntempLink;
}

//mark next to first link as first
   head = head->next;

//return the deleted link returntempLink;
}
```

Display List Operation

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

```
//display the list
voidprintList(){ 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(" ]");
}
```

```
#include <bits/stdc++.h>
using namespace std;

// A linked list node
class Node
{
	public:
int data;
Node* next;
	Node* prev;
};

/* Given a reference (pointer to pointer)  to
the head of a list
and an int, inserts a new node on the  front
of the list. */
void push(Node** head_ref, int new_data)
{
	/* 1. allocate node */
	Node* new_node = new Node();

	/* 2. put in the data */
	new_node->data = new_data;

	/* 3. Make next of new node as head
and previous as NULL */		new_node-
>next = (*head_ref);
	new_node->prev = NULL;

	/* 4. change prev of head node to new node */
if ((*head_ref) != NULL)
		(*head_ref)->prev = new_node;
```

```
        /* 5. move the head to point to the new node */
        (*head_ref) = new_node;
}

/* Given a node as prev_node, insert  a
new node after the given node */
void insertAfter(Node* prev_node, int new_data)
{
        /*1. check if the given prev_node is NULL */
        if (prev_node == NULL)
        {
                cout<<"the given previous node cannot be NULL";
        return;
        }

        /* 2. allocate new node */
        Node* new_node = new Node();

        /* 3. put in the data */
        new_node->data = new_data;

        /* 4. Make next of new node as next of prev_node */
new_node->next = prev_node->next;

        /* 5. Make the next of prev_node as new_node */
        prev_node->next = new_node;

        /* 6. Make prev_node as previous of new_node */
new_node->prev = prev_node;

        /* 7. Change previous of new_node's next node */
if (new_node->next != NULL)
                new_node->next->prev = new_node;
}

/* Given a reference (pointer to pointer) to the head  of
a DLL and an int, appends a new node at the end */
void append(Node** head_ref, int new_data)
{
        /* 1. allocate node */
        Node* new_node = new Node();

        Node* last = *head_ref; /* used in step 5*/

        /* 2. put in the data */
        new_node->data = new_data;

        /* 3. This new node is going to be the last node, so
make next of it as NULL*/
        new_node->next = NULL;

        /* 4. If the Linked List is empty, then make the new
                node as head */
if (*head_ref == NULL)
        {
                new_node->prev = NULL;
                *head_ref = new_node;
return;
        }

        /* 5. Else traverse till the last node */
        while (last->next != NULL)
                last = last->next;

        /* 6. Change the next of last node */
        last->next = new_node;

        /* 7. Make last node as previous of new node */
        new_node->prev = last;

        return;
}

// This function prints contents of  //
linked list starting from the given node
void printList(Node* node)
{
        Node* last;
        cout<<"\nTraversal in forward direction \n";
        while (node != NULL)
        {
                cout<<" "<<node->data<<" ";
last = node;
                node = node->next;
        }

        cout<<"\nTraversal in reverse direction \n";
        while (last != NULL)
        {
                cout<<" "<<last->data<<" ";
                last = last->prev;
```

```
	}
}

/* Driver program to test above functions*/
int main()
{
	/* Start with the empty list */
	Node* head = NULL;

	// Insert 6. So linked list becomes 6->NULL
	append(&head, 6);

	// Insert 7 at the beginning. So
// linked list becomes 7->6->NULL
	push(&head, 7);

	// Insert 1 at the beginning. So
// linked list becomes 1->7->6->NULL
push(&head, 1);

	// Insert 4 at the end. So linked
// list becomes 1->7->6->4->NULL
	append(&head, 4);

	// Insert 8, after 7. So linked	//
list becomes 1->7->8->6->4->NULL
	insertAfter(head->next, 8);

	cout << "Created DLL is: ";
	printList(head);

	return 0;
}
```

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} intlastNode = 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). intGetNth(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.

Question No. 4:

You are given a circular doubly linked list that contains integers as the data in each node. These data on each node is distinct. You have developed a special algorithm that prints the three continuous elements of the list, starting from the first element or head of the list and runs for infinite time. For example, if the list is {1,9,12,7}, then the output of the algorithm will be {1,9,12,9,12,7,12,7,1,...}. The output contains the infinite number of elements because it is a circular list.

You are given only a part of the output that has been returned by the algorithm. Your task is to determine the number of elements available in the original list and print the respective elements.

Note

- It is guaranteed that the provided part of the output of the algorithm is sufficient enough to calculate the size of the original list.
- Please read the sample explanation carefully and use the following definition of the doubly linked list:

```
class    Node    {
Object data;
  Node next;
  Node prev;
}
```

Input format

- First line: Integer N that denotes the length of the list which is returned as the output of the algorithm
- Next line: N space-separated integers that denote the elements of the list which is returned as the output of the algorithm

Output format

Your task is to print the output in the following format:

- First line: Length of the original list
- Second line: Space-separated integers that denote the elements of the original list