NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
DATA STRUCTURES LAB

Lab Session 06

Instructors: 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.

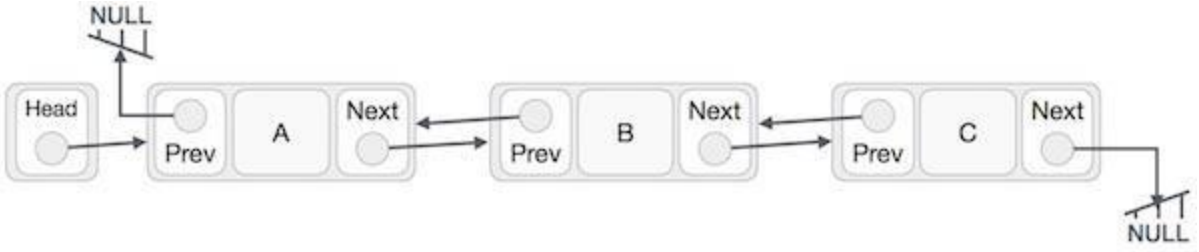
```
struct Node
{
    char name[20];      // Name of up to 20 letters
    int age;            // D.O.B. would be better
    float height;       // In meters
    Node *next;         // Pointer to next node
    Node *previous;     // Pointer to previous node
};
```

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

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

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.
4) 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

Every node of DLL Require extra space for a previous pointer.
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.

Circular Linked List:

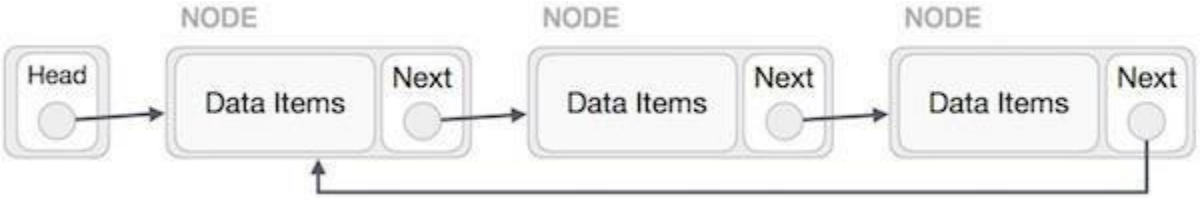
Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list. The elements points to each other in a circular way which forms a circular chain. The circular linked list has a dynamic size which means the memory can be allocated when it is required.

Application of Circular Linked List

- The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running. All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed.
- Another example can be Multiplayer games. All the Players are kept in a Circular Linked List and the pointer keeps on moving forward as a player's chance ends.
- Circular Linked List can also be used to create Circular Queue. In a Queue we have to keep two pointers, FRONT and REAR in memory all the time, where as in Circular Linked List, only one pointer is required.

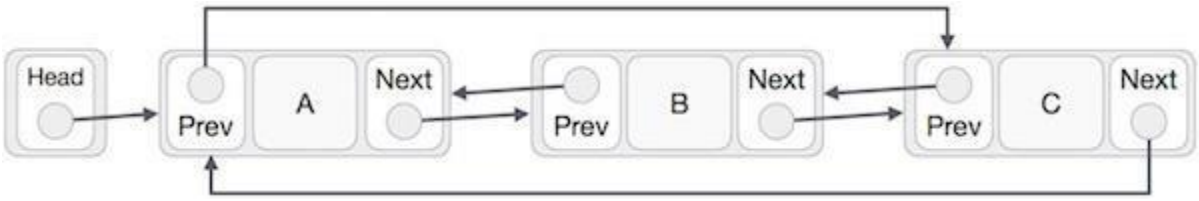
Singly Linked List as Circular

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.



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

```
class Node {
  public:
  int data;
  //pointer to the next node
  node* next;

  node() {
    data = 0;
    next = NULL;
```

```
  }

  node(int x) {
    data = x;
    next = NULL;
  }
};
class CircularLinkedList {
  public:
  node *head;
  //declaring the functions

  //function to add Node at front
  int addAtFront(node *n);
  //function to check whether Linked list is empty
  int isEmpty();
  //function to add Node at the End of list
  int addAtEnd(node *n);
  //function to search a value
  node* search(int k);
  //function to delete any Node
  node* deleteNode(int x);

  CircularLinkedList() {
    head = NULL;
  }
}
int CircularLinkedList :: addAtFront(node *n) {
  int i = 0;
  /* If the list is empty */
  if(head == NULL) {
    n->next = head;
    //making the new Node as Head
    head = n;
    i++;
  }
  else {
    n->next = head;
    //get the Last Node and make its next point to new Node
    Node* last = getLastNode();
    last->next = n;
    //also make the head point to the new first Node
    head = n;
    i++;
  }
  //returning the position where Node is added
  return i;
} int CircularLinkedList :: addAtEnd(node *n) {
```

```
  //If list is empty
  if(head == NULL) {
    //making the new Node as Head
    head = n;
    //making the next pointer of the new Node as Null
    n->next = NULL;
  }
  else {
    //getting the last node
    node *last = getLastNode();
    last->next = n;
    //making the next pointer of new node point to head
    n->next = head;
  }
}

node* CircularLinkedList :: search(int x) {
  node *ptr = head;
  while(ptr != NULL && ptr->data != x) {
    //until we reach the end or we find a Node with data x, we keep moving
    ptr = ptr->next;
  }
  return ptr;
}
node* CircularLinkedList :: deleteNode(int x) {
  //searching the Node with data x
  node *n = search(x);
  node *ptr = head;
  if(ptr == NULL) {
    cout << "List is empty";
    return NULL;
  }
  else if(ptr == n) {
    ptr->next = n->next;
    return n;
  }
  else {
    while(ptr->next != n) {
      ptr = ptr->next;
    }
    ptr->next = n->next;
    return n;
  }
}
```

Exercise:

Question No. 1:

Menu driven program for all operations on **double linked list**
Operations to be performed:

- **traverse()**: To see the contents of the linked list, it is necessary to traverse the given linked list.
- The given traverse() function traverses and prints the content of the linked list.
- **insertAtFront():** This function simply inserts an element at the front/beginning of the linked list.
- **insertAtEnd():** This function inserts an element at the end of the linked list.
- **insertAtPosition():** This function inserts an element at a specified position in the linked list.
- **deleteFirst():** This function simply deletes an element from the front/beginning of the linked list.
- **deleteEnd():** This function simply deletes an element from the end of the linked list.
- **deletePosition():** This function deletes an element from a specified position in the linked list.
- **maximum()**: This function finds the maximum element in a linked list.
- **mean():** This function finds the mean of the elements in a linked list.
- **sort():** This function sort the given linked list in ascending order.
- **reverseLL():** This function reverses the given linked list.

Question No. 2:

Given a double linked list, find the middle of the linked list. For example, if the given linked list is 1->2->3->4->5 then the output should be 3.
If there are even nodes, then there would be two middle nodes, we need to print the second middle element. For example, if given linked list is 1->2->3->4->5->6 then the output should be 4.

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

Question No. 5:

Write a removeDuplicates() function that takes a list and deletes any duplicate nodes from the list. The list is not sorted. For example if the linked list is 12->11->12->21->41->43->21 then removeDuplicates() should convert the list to 12->11->21->41->43.