DIPLOMA WALLAH

(Your **One Stop Hub** For Diploma Resources)

DATA STRUCTURES

WITH PYTHON

Complete Notes Based on Full Syllabus

- Diploma Engineering

4th Semester

© Diploma Wallah. All Rgrts Reserved

Unauthorized sharing/selling is strictly prohibited

www.diplomawallah.in

Notes prepared by **Sangam**

Diploma Wallah

Sangam

unit-07

Doubly linked list (DLL)

A Doubly linked list (DLL) is a data structure consisting of nodes where:

- Each node contains:
  - a. Data
  - b. pointer to the next node (next)
  - c. pointer to the previous node. (prev)
- Navigation is possible in both directions - forward and backward.

Why DDL (Double linked list)

- unlike single linked list (which can only be traversed forward), DLL allows backward traversal as well.
- Insertion and deletion can be more efficient for some cases.

Ex-

1. image viewer
   - Clicking 'Next' → move forward in the DLL.
   - Clicking "previous" - move backward in the DLL.

Structure:

[prev] <--> [Data] <--> [Next]

Ex - of 3 node.

NULL <- 10 <-> 20 <-> 30 -> NULL

Diplomawallah.in

4. Basic operations in DLL

1. Creating Nodes.

```
Class Node:
    def __init__(self, data):
        self.data = data  # Store node value
        self.prev = None  # address of prev node
        self.next = None  # "  " next node
```

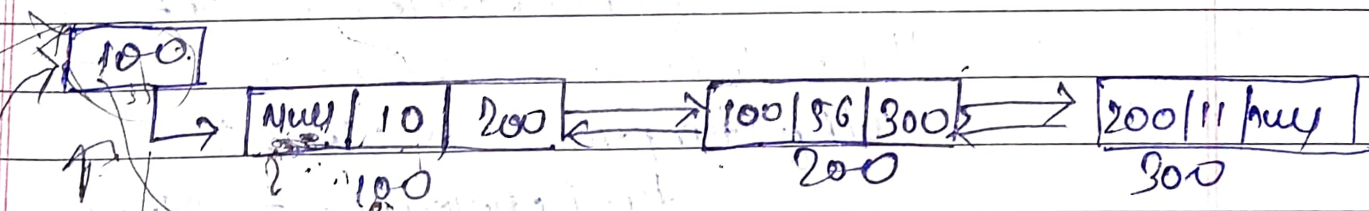
Creating the DLL and Traversing

```
Class Double:
  def __init__(self):
      self.head = None
  # insert at end
  def append(self, data):
      new_node = Node(data)  # new node
      if self.head is None:
          self.head = new_node
          return
      temp = self.head
      while temp.next:
          temp = temp.next
      temp.next = new_node  # in the next of last node we put next node
      new_node.prev = temp
                  # ↳ and new node prev is last node
```

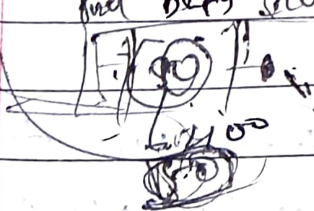
Date ______
Page ______

```
#forward traversal
def display_forward(self):
    temp = self.head
    while temp:  # till temp == none.
        print(temp.data, end="<->")
        temp = temp.next  # move to next node.
    print("NULL")  # show null in last
#Backward traversal
def display_backward(self):
    temp = self.head
    if not temp:
        return
    while temp.next:
        temp = temp.next
    while temp:
        print(temp.data, end="<-->")
        temp = temp.prev
    print("NULL")
```

Diplomawallah.in



logical structure.



Searching for a node

```
def search(self, key):
    temp = self.head
    while temp:
        if temp.data == key:
            return True
        temp = temp.next
    return False
```

* Deleting a node

```
def delete(self, key):
    temp = self.head
    while temp:
        if temp.data == key:
        # Case 1: Deleting head node
         if temp.prev is None:
             self.head = temp.next
         if self.head:
             self.head.prev = None
        # Case 2: Deleting last node
         elif temp.next is None:
             temp.prev.next = None
        # Case 3: Middle Node
         else:
             temp.prev.next = temp.next
             temp.next.prev = temp.prev
         return
        temp = temp.next
```

Diplomawallah.in

Ex

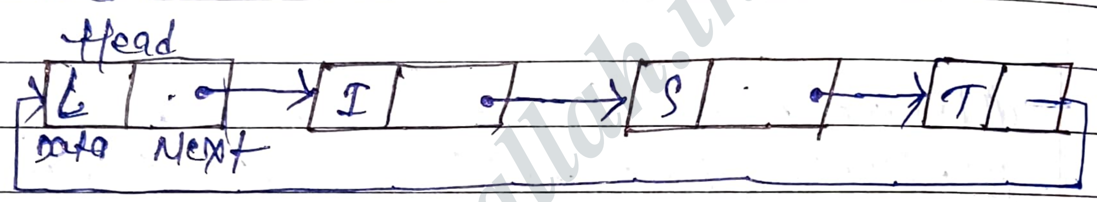
```
dll = Double()
dll.append(10)
dll.append(20)
dll.append(30)

dll.display_forward()        # 10 <--> 20 <--> 30 <--> Null
dll.display_backward()       # 30 <--> 20 <--> 10 <--> Null

dll.prepend(5)
dll.display_forward()        # 5 <--> 10 <--> 20 <--> 30 <--> Null

print(dll.search(20))        # True
print(dll.search(99))        # False

dll.delete(20)
dll.display_forward()        # 5 <--> 10 <--> 30 <--> Null
```

Diplomawallah.in

NULL 7 (w)
head = new node
new.D = head

Circular Linked List

A circular linked list is a linked list where the last node points back to the first node, forming a circle.
Unlike a normal singly linked list where the last node points to None (null) here the last node's next pointer points to the head node.



- It allows continuous traversal without stopping.
- Useful for applications like round-robin scheduling, buffering etc.

- Appending nodes in a circular linked list

1. Create a new node with given data
2. If the list is empty (no nodes yet):
   - Set the new node's next pointer to point itself.
   - Make this new node the head of the list.
3. If the list is not empty
   - Traverse the list to find the last node (the node whose next points to head).
   - Set the last node's next pointer to the new node
   - Set the new node's next pointer to the head node.

Diplomawallah.in

Date ____
Page ____

Q)
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class circularlinkedlist:
    def __init__(self):
        self.head = None
    def append(self, data):
        new_node = Node(data)
        if not self.head:
            # list is empty: new node point to itself.
            self.head = new_node
            new_node.next = new_node
        else:
            # find last node (whose next points to
              head)
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            # Append new node at the end
            temp.next = new_node
            new_node.next = self.head
    def display(self):
        if not self.head:
            print("List is empty.")
            return
```

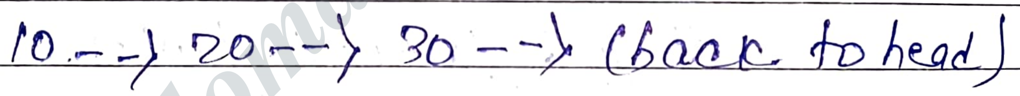
Diplomawallah.in

```
temp = self.head
while True:
    print(temp.data, end = " -> ")
    temp = temp.next
    if temp == self.head:
print("(back to head)")
```

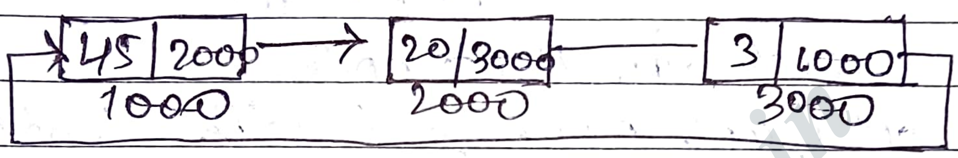
Ex

```
cll = CircularLinkedList()
cll.append(10)
cll.append(20)
cll.append(30)
cll.display()
```

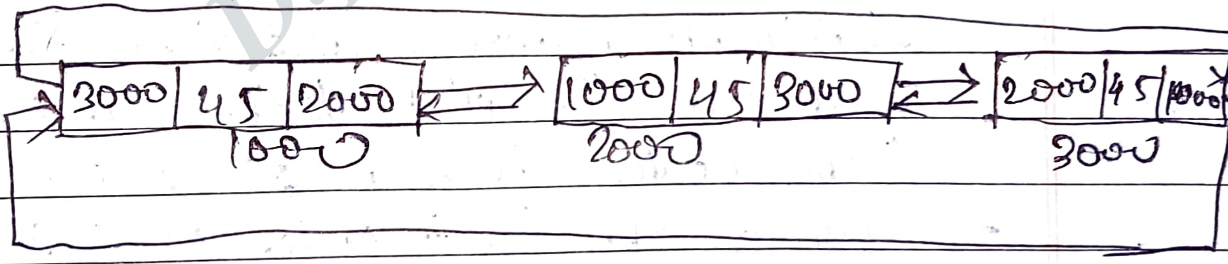
O/P ->



* circular singly linked list
circular singly linked list similar to the singly linked list except that the last node of the circular singly linked list points to the first node.



* Circular doubly linked list
Circular doubly linked list is similar to the doubly linked list except that the last node of circular doubly linked list points to the first node and the first node points to the last node.



Diplomawallah.in

Algorithm to create circular singly linked list

1. Define a node structure
- each node contains two parts:
  (a) data → to store value
  (b) next → to store the address of the next node.

2. Create the first node
- Allocate memory for the first node.
- Store the data value in it.
- make its next pointer point to itself
(because circular list, the last node always points to the first node.)
- This node will be head node.

3. Create additional nodes.
- for each new node:
  - Allocate memory and store the data value
  - Traverse the list to the last node
  (the node whose next point back to head)
  - Change the last node next to the new node
  - make the new node's next point back to the head.

4. update the link properly.

5. Traverse a circular list, start from the head.
Keep moving to next nodes and processing their data.

Diplomawallah.in