香港中文大學（深圳）
The Chinese University of Hong Kong

CSC3100 Data Structures
Lecture 7: List

Li Jiang
School of Data Science (SDS)
The Chinese University of Hong Kong, Shenzhen

Outline

- List and List ADT
- Four types of linked lists
  - Singly linked list
  - Doubly linked list
  - Circular linked list
  - Doubly circular linked list

List

- Definition in Wikipedia:
  - A list is an abstract data type (ADT) that represents a finite number of ordered values, where the same value may occur more than once

- A list: $a_1, a_2, a_3, \ldots, a_N$
  - We say that the size of this list is N
  - For any list except the empty list, we say:
    - The first element of the list is a_1, and the last element is a_N
    - a_{i+1} follows (succeeds) a_i $(i < N)$
    - a_{i-1} precedes a_i $(i > 1)$
    - The predecessor of a_1 or the successor of a_N are not defined

List ADT

- Some popular operations on List ADT are:
  - printList
  - makeEmpty
  - Find
    - return the position of the first occurrence of a key, e.g., given the list: 34, 12, 52, 16, 12, find(52) returns 3
  - insert
    - insert some key at some position, e.g., insert(X, 3)
  - delete
    - delete some key from some position, e.g., delete(52)
  - next & previous (optional)

Why list?

- An array stores data with the following limitations:
  - The size of the array is fixed, so we must know the upper limit on the number of elements in advance
  - Inserting a new element in an array is expensive because the room has to be created for the new elements and existing elements have to be shifted

- For example, if we maintain a sorted list of IDs in an array id[ ] = [1000, 1010, 1050, 2000, 2040]
  - If we insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000)
  - If we want to delete 1010, everything after 1010 has to be moved

Advantages

- Dynamic data structure:
  - The size of a linked list is not fixed as it can vary arbitrarily
- Insertion and deletion are easier:
  - In linked lists, insertion and deletion are easier than those on arrays, since the elements of an array are stored in a consecutive location, while the elements of a linked list are stored in a random location
  - If we want to insert or delete the element in an array, then we need to shift the elements for creating the space, while in a linked list, we do not have to shift the elements
- Memory efficient:
  - Its memory consumption is efficient as the size of the linked list can grow or shrink according to our requirements

Disadvantages

- Memory usage:
  - The node in a linked list occupies more memory than array as each node has one simple variable and one pointer variable that occupies 4 bytes in memory
- Traversal:
  - In an array, we can randomly access the element by index, while in a linked list, the traversal is not easy, i.e., if we want to access the element in a linked list, we cannot access the element randomly
- Reverse traversing:
  - In a linked list, backtracking or reverse traversing is difficult
  - In a doubly linked list, it is easier but requires more memory

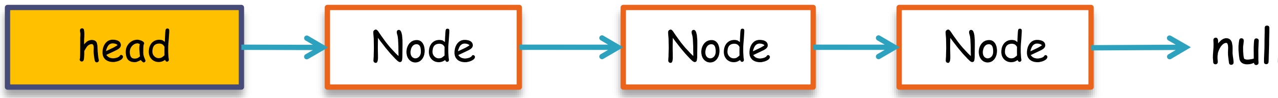
Types of linked list

- We study four types of linked lists
  - Singly linked list
  - Doubly linked list
  - Circular linked list
  - Doubly circular linked list

Lists [edit]

- Doubly linked list
- Array list
- Linked list
- Association list
- Self-organizing list
- Skip list
- Unrolled linked list
- VList
- Conc-tree list
- Xor linked list
- Zipper
- Doubly connected edge list also known as half-edge
- Difference list
- Free list

Singly linked list



- Consists of a series of nodes (Node class)
- A singly linked list, each node is composed of data and a pointer

```
class Node {
    int element;
    Node next;
}
```

- Must know the head for keeping track of it
- Not necessarily adjacent in memory
- Flexible on element insertion and deletion

```
// constructor
public Node(int x) {
    element = x;
    next = null;
}
```

Operations on singly linked list

Insertion

The insertion into a singly linked list can be performed at different positions. Based on the position of the new node being inserted, the insertion is categorized into the following categories.

SN	Operation	Description
1	Insertion at beginning	It involves inserting any element at the front of the list. We just need to a few link adjustments to make the new node as the head of the list.
2	Insertion at end of the list	It involves insertion at the last of the linked list. The new node can be inserted as the only node in the list or it can be inserted as the last one. Different logics are implemented in each scenario.
3	Insertion after specified node	It involves insertion after the specified node of the linked list. We need to skip the desired number of nodes in order to reach the node after which the new node will be inserted. .

Operations on singly linked list

Deletion and Traversing

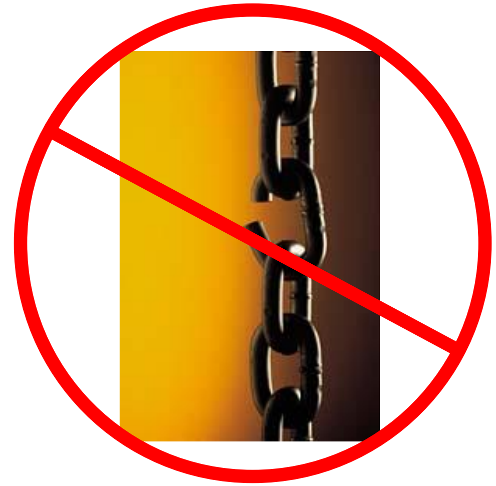
The Deletion of a node from a singly linked list can be performed at different positions. Based on the position of the node being deleted, the operation is categorized into the following categories.

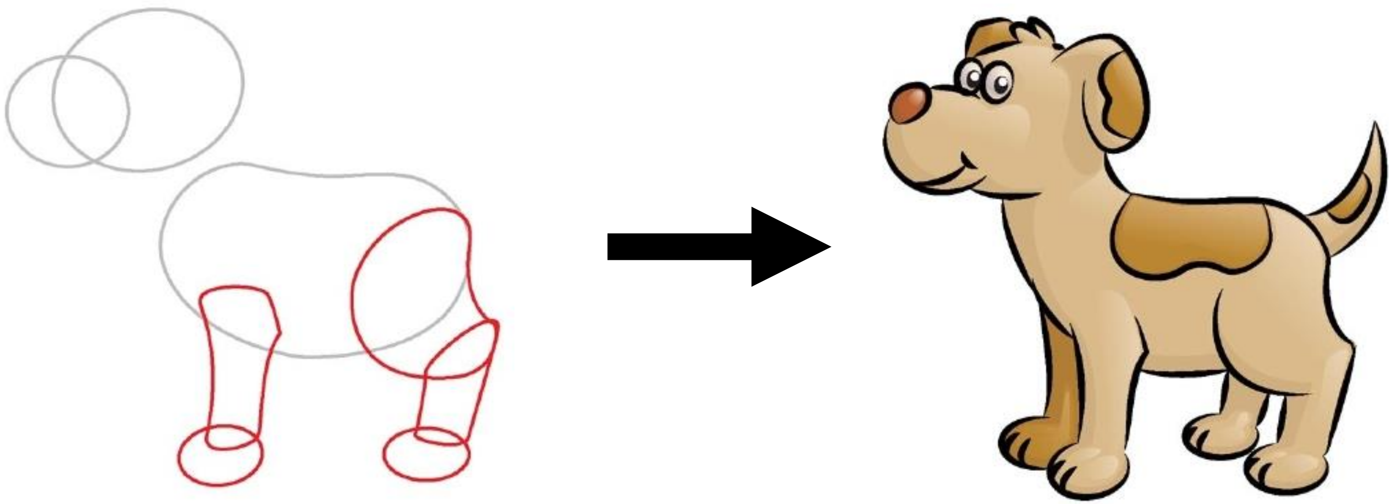
SN	Operation	Description
1	Deletion at beginning	It involves deletion of a node from the beginning of the list. This is the simplest operation among all. It just need a few adjustments in the node pointers.
2	Deletion at the end of the list	It involves deleting the last node of the list. The list can either be empty or full. Different logic is implemented for the different scenarios.
3	Deletion after specified node	It involves deleting the node after the specified node in the list. we need to skip the desired number of nodes to reach the node after which the node will be deleted. This requires traversing through the list.
4	Traversing	In traversing, we simply visit each node of the list at least once in order to perform some specific operation on it, for example, printing data part of each node present in the list.
5	Searching	In searching, we match each element of the list with the given element. If the element is found on any of the location then location of that element is returned otherwise null is returned. .

Linked list: insert & delete

- Must be careful not to break the chain!
- Consider the special cases

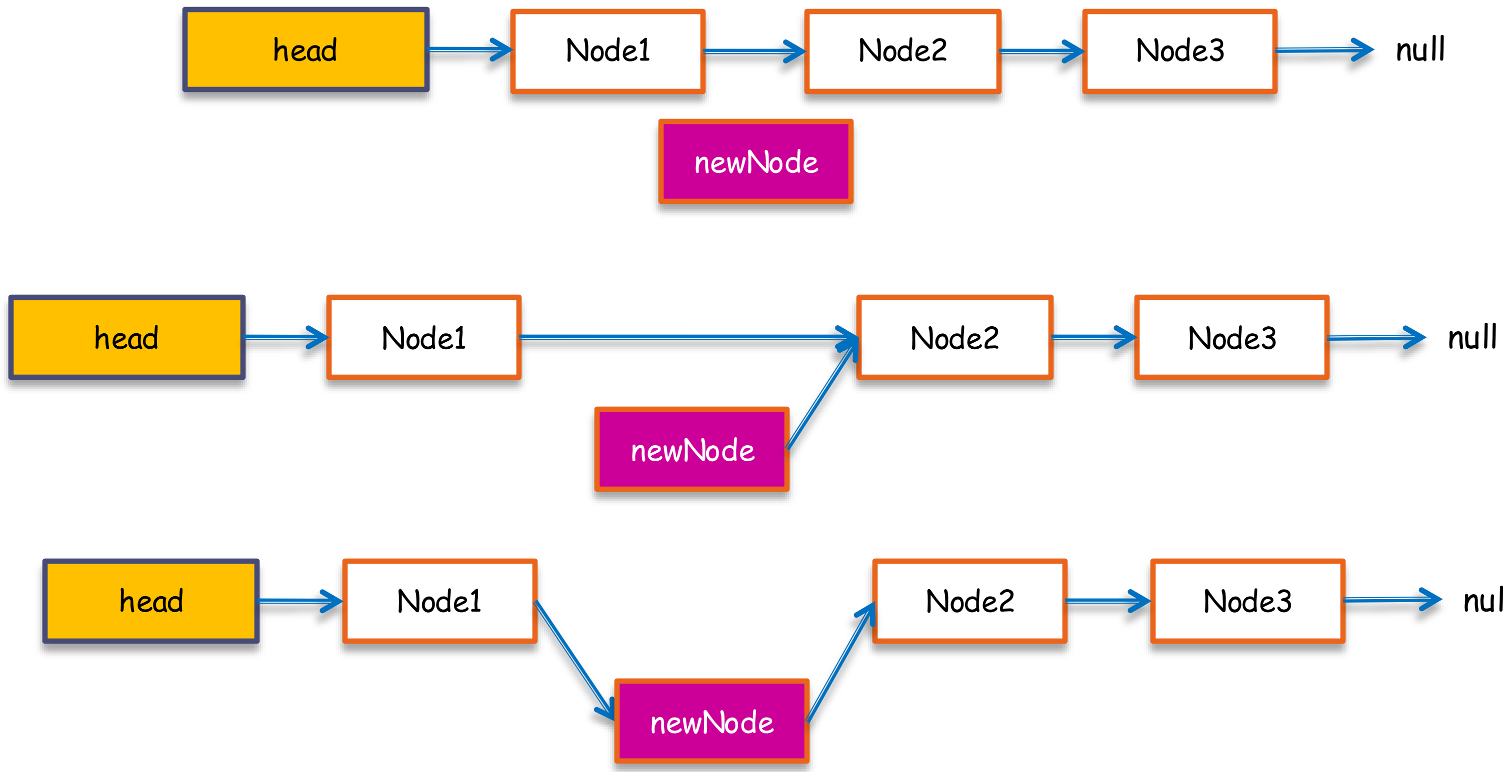
- Draw a picture before any coding!





Linked list: insert

- Add a new element to the list



Linked list: insert

```
void insert(int x, int p) {
        Node tmpNode = new Node(x);
        Node prevNode = head;

        if (p == 0) {
                tmpNode.next = head;
                head = tmpNode;
                return;
        }

        for (int i=0; i<p-1; i++) {
                if (prevNode == null)
                        break;
                prevNode = prevNode.next;
        }
        if(prevNode == null) return;

        tmpNode.next = prevNode.next;
        prevNode.next = tmpNode;
}
```

- *tmpNode* is a new node that contains x
- *prevNode* will be used for finding the previous node of *tmpNode*

Handle the situation: No previous node before tmpNode

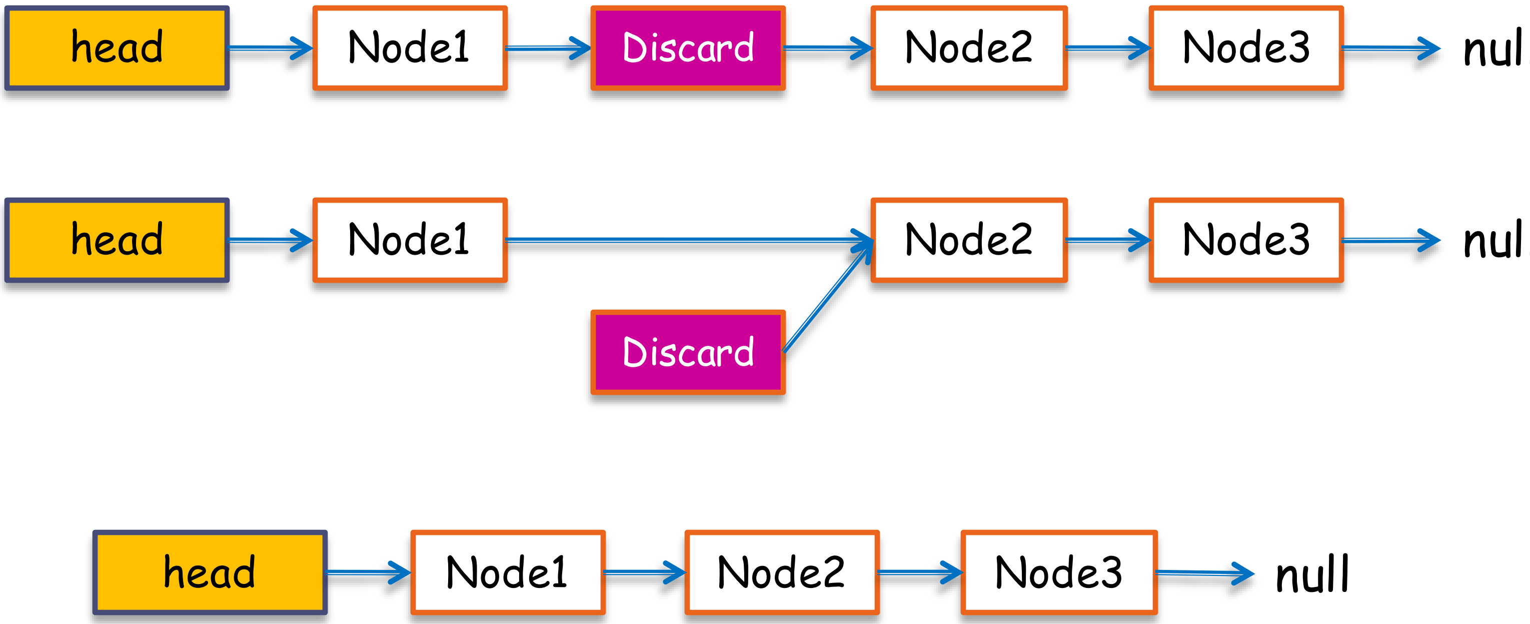
Moves to the position p-1 or the end of the list if p is larger than the size of list

Link the new node

What are the time complexities of the best/worst/average cases?

Linked list: delete

- Delete a node from the list



Linked list: delete

```
void delete(int p) {
    if(p == 0 && head != null)
        head = head.next;
        return;

    Node prevNode = head;
    for(int i=0; i<p-1; i++)
            if (prevNode == null)
                    break;
            prevNode = prevNode.next;

    if(prevNode == null || prevNode.next == null) return;

    Node targetNode = prevNode.next;
    if (targetNode != null)
            prevNode.next = targetNode.next;
}
```

Handle the situation: No previous node before targetNode

Go to the specific position

No node at the position

Bypass the target node

What are the time complexities of the best/worst/average cases?

Question

- Given an array A[ ] with n elements, how to insert a new element x into it?
  - If the size of A[ ] is larger than n, insert x directly
  - If the size of A[ ] is equal to n, how to do it?

- Given an array B[ ], how to delete an element x from it?

Applications of singly linked list

- It is used to implement stacks and queues, which are fundamental data structures
- To prevent the collision between the data in the hash map, we use a singly linked list
- A casual notepad uses a singly linked list to perform undo/redo functions
- …
- More detailed examples will be given in next lecture

Java implementation

```java
class Node{
    public int data;
    public Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }

    public void displayNodeData() {
        System.out.println("{ " + data + " } ");
    }
}

```

Java implementation

```java
public class SinglyLinkedList {
    private Node head;

    public boolean isEmpty() {
        return (head == null);
    }

    // used to insert a node at the start of linked list
    public void insertFirst(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }

    // used to insert a node at the last of linked list
    public void insertLast(int data) {
        Node newNode = new Node(data);
        if(head == null) {
            head = newNode;
        }
        else {
            Node current = head;
            while(current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }

```

Java implementation

```java
    // used to delete node from the start of linked list
    public void deleteFirst() {
        if(head != null) {
            head = head.next;
        }
    }

    // used to delete node after the node whose value is 'data'
    public void deleteAfter(int data) {
        if(head == null) return;

        Node temp = head;
        while(temp.next != null && temp.data != data) {
            temp = temp.next;
        }
        if(temp.next != null) {
            temp.next = temp.next.next;
        }
    }

    // for printing linked list
    public void printLinkedList() {
        System.out.println("Printing LinkedList (head --> last) ");
        Node current = head;
        while(current != null) {
            current.displayNodeData();
            current = current.next;
        }
        System.out.println();
    }
}
```

Java implementation

LinkedListMain.java

```java
package list;

public class LinkedListMain {

    public static void main(String[] args) {
        SinglyLinkedList myLinkedList = new SinglyLinkedList();

        myLinkedList.insertFirst(5);
        myLinkedList.insertFirst(6);
        myLinkedList.insertFirst(7);
        myLinkedList.insertFirst(1);
        myLinkedList.insertLast(2);
        // 1->7->6->5->2


        myLinkedList.deleteAfter(1);
        // 1->6->5->2

        myLinkedList.printLinkedList();
    }

}
```

When you run above program, you will get below output:

```
Printing LinkedList (head --> last)
{ 1 }
{ 6 }
{ 5 }
{ 2 }
```

Exercise 1: how to get the middle node?

- Given a linked list with its head node known, write the java codes to find the middle node

```java
// find middle element in linked list
public Node findMiddleNode() {
    Node slowPointer, fastPointer;
    slowPointer = fastPointer = head;

    while(fastPointer != null && fastPointer.next != null && fastPointer.next.next != null) {
        fastPointer = fastPointer.next.next;
        slowPointer = slowPointer.next;
    }
    return slowPointer;
}
```

Exercise 2: how to reverse a linked list?

- Given a linked list with its head node known, write the java codes to reverse the linked list

```java
// an iterative solution to reverse a linked list
public void reverseLinkedList() {
    Node currentNode = head;
    // for first node, previous node will be null
    Node previousNode = null;
    Node nextNode;
    while(currentNode != null) {
        nextNode = currentNode.next;
        // reverse the link
        currentNode.next = previousNode;
        // move current node and previous node by 1 node
        previousNode = currentNode;
        currentNode = nextNode;
    }
    head = previousNode;
}
```

```java
public static void main(String[] args) {
    SinglyLinkedList myLinkedList = new SinglyLinkedList();

    myLinkedList.insertFirst(5);
    myLinkedList.insertFirst(6);
    myLinkedList.insertFirst(7);
    myLinkedList.insertFirst(1);
    myLinkedList.insertLast(2);
    // 1->7->6->5->2
    myLinkedList.printLinkedList();


    myLinkedList.reverseLinkedList();
    // 2->5->6->7->1
    System.out.println("After reversing");
    myLinkedList.printLinkedList();
}
```

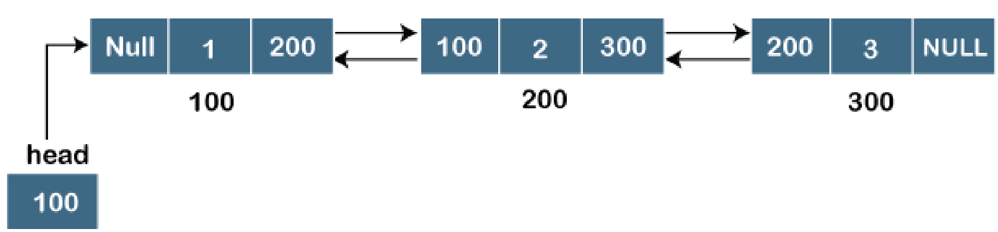
output:

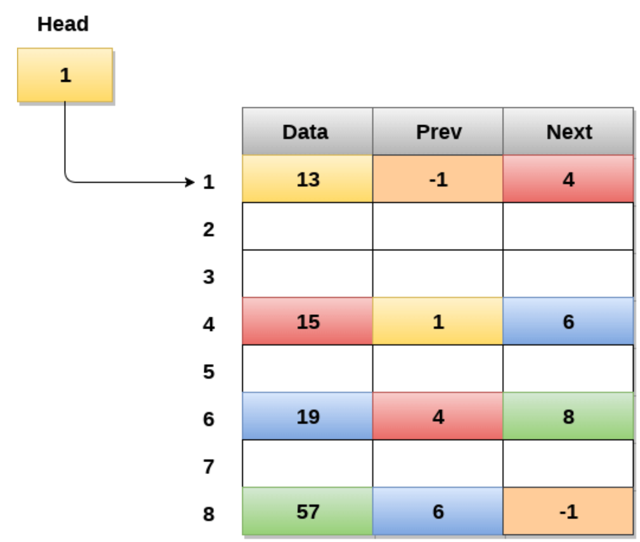
```
Printing LinkedList (head --> last)
{ 1 }
{ 7 }
{ 6 }
{ 5 }
{ 2 }

After reversing
Printing LinkedList (head --> last)
{ 2 }
{ 5 }
{ 6 }
{ 7 }
{ 1 }
```

Doubly linked list

- A doubly linked list has three parts in a node
  - A data part
  - A pointer to its previous node
  - A pointer to its next node

```
class Node {
    int element;
    Node next;
    Node prev;
}
```





Memory Representation of a Doubly linked list

Operations on doubly linked list

SN	Operation	Description
1	Insertion at beginning	Adding the node into the linked list at beginning.
2	Insertion at end	Adding the node into the linked list to the end.
3	Insertion after specified node	Adding the node into the linked list after the specified node.
4	Deletion at beginning	Removing the node from beginning of the list
5	Deletion at the end	Removing the node from end of the list.
6	Deletion of the node having given data	Removing the node which is present just after the node containing the given data.
7	Searching	Comparing each node data with the item to be searched and return the location of the item in the list if the item found else return null.
8	Traversing	Visiting each node of the list at least once in order to perform some specific operation like searching, sorting, display, etc.

Applications of doubly linked list

- Doubly linked list is used in navigation systems, for front and back navigation (e.g., back and next functions in the browser)

- It is easily possible to implement other data structures like a binary tree, hash tables, stack, etc.

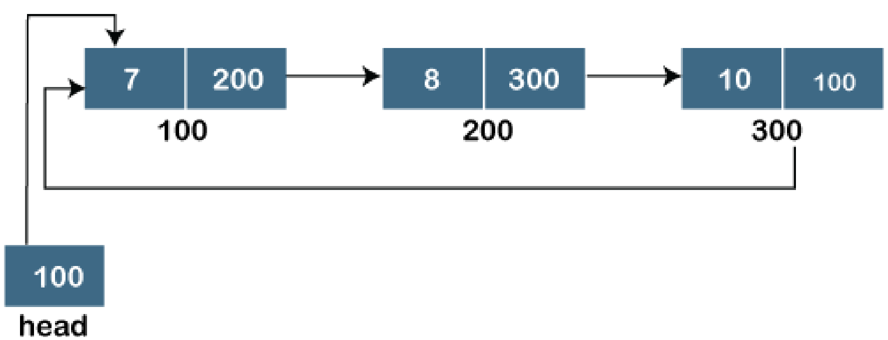
- It is used in music playing system where you can easily play the previous one or next one song as many times one person wants to

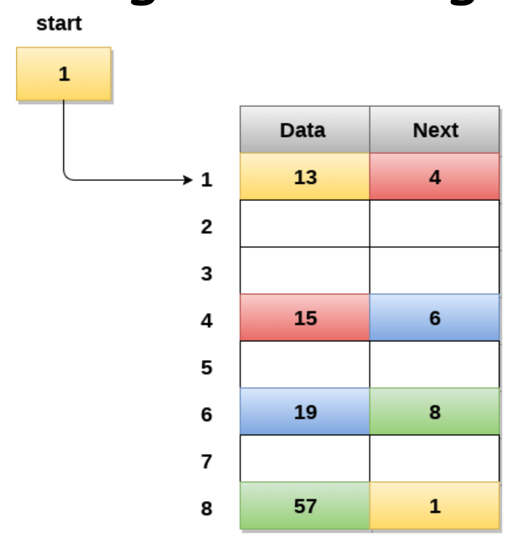
- In operating systems, the thread scheduler maintains a doubly-linked list of all the running processes
  - It is easy to move a process from one queue into another queue

Circular linked list

- It is a variation of a singly linked list

- Singly linked list vs a circular linked list
  - In a singly linked list, the last node does not point to any node
  - In a circular linked list, the last node links to the first node
  - The circular linked list has no starting and ending node, so we can traverse in any direction





Memory Representation of a circular linked list

Operations on circular linked list

Insertion

SN	Operation	Description
1	Insertion at beginning	Adding a node into circular singly linked list at the beginning.
2	Insertion at the end	Adding a node into circular singly linked list at the end.

Deletion & Traversing

SN	Operation	Description
1	Deletion at beginning	Removing the node from circular singly linked list at the beginning.
2	Deletion at the end	Removing the node from circular singly linked list at the end.
3	Searching	Compare each element of the node with the given item and return the location at which the item is present in the list otherwise return null.
4	Traversing	Visiting each element of the list at least once in order to perform some specific operation.

Applications of circular linked list

- It is used by the operating system to share time for different users, generally using a Round-Robin time-sharing mechanism
  - Each user gets an equal share of something in turns
  - It is the oldest, simplest scheduling algorithm, which is mostly used for multitasking

- Multiplayer games use a circular list to swap between players in a loop

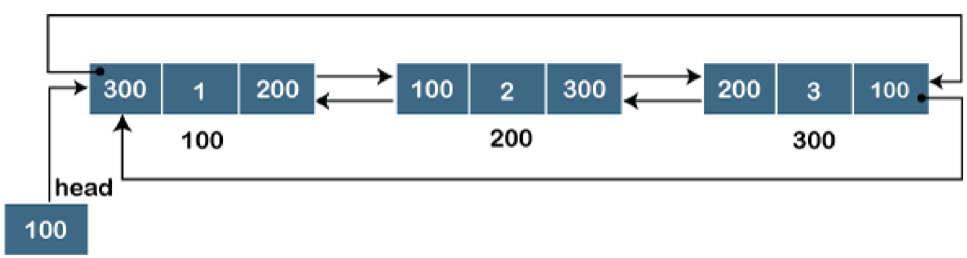
- It can also be used to implement queues by maintaining a pointer to the last inserted node and the front can always be obtained as next of last

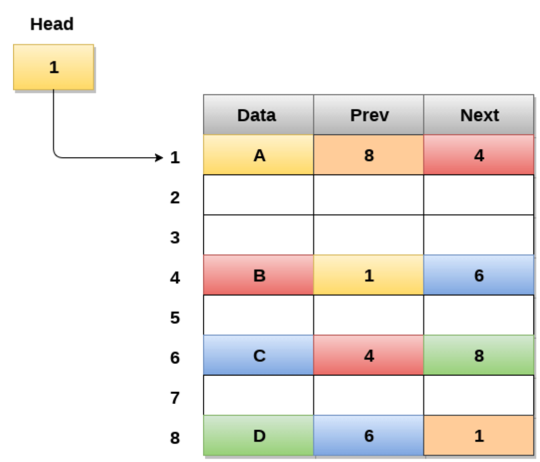
Question

- Both doubly linked list and circular linked list can overcome the limitation of singly linked list, but which one is better?
  - Hint: think the time and space cost
  - There is a trade-off between time and space

Doubly circular linked list

- It combines circular linked list and doubly linked list
  - The last node is linked to the first node and creates a circle
  - Each node holds the address of the previous node
- It has three parts in a node
  - Two address parts
  - One data part
  - Similar to the doubly linked list





Memory Representation of a Circular Doubly linked list

Operations on doubly circular linked list

SN	Operation	Description
1	Insertion at beginning	Adding a node in circular doubly linked list at the beginning.
2	Insertion at end	Adding a node in circular doubly linked list at the end.
3	Deletion at beginning	Removing a node in circular doubly linked list from beginning.
4	Deletion at end	Removing a node in circular doubly linked list at the end.

Traversing and searching in circular doubly linked list is similar to that in the circular singly linked list.

Recommended reading

- Reading
  - Chapter 10, textbook
- Next lecture
  - Applications of list