

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

# CSC3100 Data Structures

## Lecture 7: List

Yixiang Fang  
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:  $A_1, A_2, A_3, \dots, A_N$ 
  - 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  $A_1$
    - The last element is  $A_N$
    - $A_{i+1}$  follows (succeeds)  $A_i$  ( $i < N$ )
    - $A_{i-1}$  precedes  $A_i$  ( $i > 1$ )
    - We will not define the predecessor of  $A_1$ , or the successor of  $A_N$



# List ADT

---

- ▶ Some popular operations on List 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?

---

- ▶ Arrays stores linear data with the following limitations:
  - The size of the arrays is fixed. We must know the upper limit on the number of elements in advance. The allocated memory is equal to the upper limit irrespective of the usage.
  - Inserting a new element in an array is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted.
- ▶ For example, in a system, if we maintain a sorted list of IDs in an array  $id[] = [1000, 1010, 1050, 2000, 2040]$ .
  - If we want to 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 in  $id[]$ , 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:
  - Insertion and deletion in linked lists are easier than those on arrays as the elements in an array are stored in a consecutive location, while in a linked list, the elements 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 occupies two types of variables, i.e., one is a simple variable, and the other is a 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 to store the back pointer



# Types of linked list

---

- ▶ We focus on linked list
  - 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



# Linked list



- ▶ Consists of a series of nodes (Node class)
- ▶ A singly linked list, each node is composed of data and a pointer
- ▶ Not necessarily adjacent in memory
- ▶ Flexible on element insertion and deletion

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

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

## Note:

"*head*" here is a node that does NOT store data



# 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

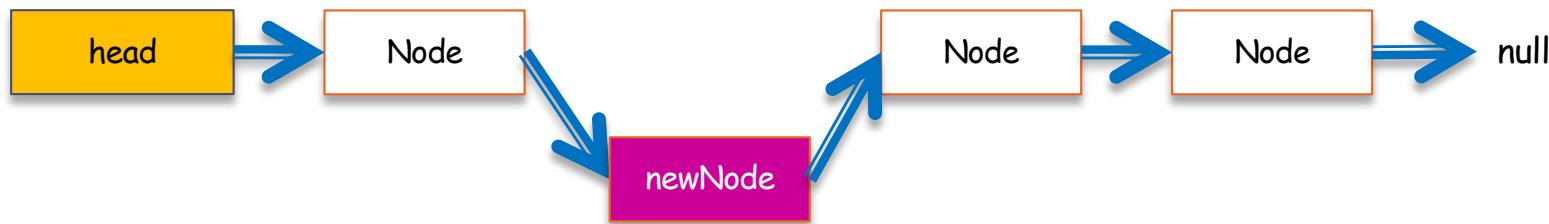
- Add a new element to the list



newNode



newNode



newNode



# Linked list :: insert

```
void insert(int x, int p) {
```

```
    Node tmpNode = new Node(x);  
    Node prevNode = head;
```

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

```
    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*

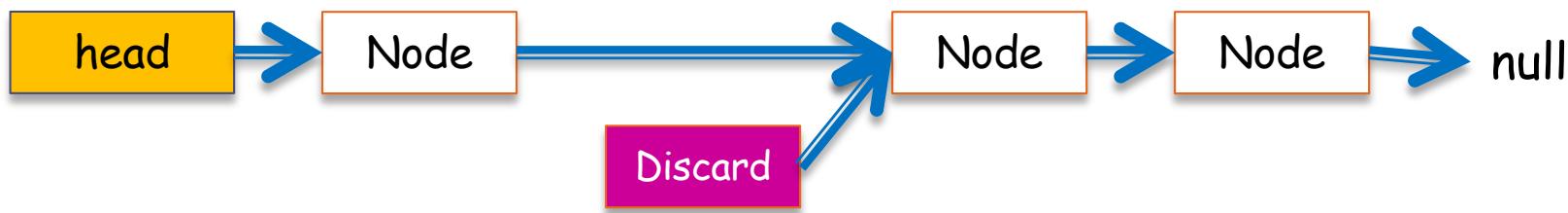
Moves to the end of the list if *p* is larger than the size of list

Link the new node



# Linked list :: delete

- ▶ Delete a node from the list





# Linked list :: delete

```
void delete(int p) {  
    Node prevNode = head;  
  
    for(int i=0; i<p-1; i++)  
        if (prevNode.next == null)  
            break;  
        else  
            prevNode = prevNode.next;  
  
    Node targetNode = prevNode.next;  
  
    if (targetNode != null)  
        prevNode.next = targetNode.next;  
}
```

Go to the specific position

Bypass the target node



# 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 like fundamental needs throughout computer science
- ▶ 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

```
1  class Node {
2      public int data;
3      public Node next;
4
5      public void displayNodeData() {
6          System.out.println("{ " + data + " } ");
7      }
8  }
9
10 public class SinglyLinkedList {
11     private Node head;
12
13     public boolean isEmpty() {
14         return (head == null);
15     }
16
17     // used to insert a node at the start of linked list
18     public void insertFirst(int data) {
19         Node newNode = new Node();
20         newNode.data = data;
21         newNode.next = head;
22         head = newNode;
23     }
24
25     // used to delete node from start of linked list
26     public Node deleteFirst() {
27         Node temp = head;
28         head = head.next;
29         return temp;
30     }
}
```

```
32     // Use to delete node after particular node
33     public void deleteAfter(Node after) {
34         Node temp = head;
35         while (temp.next != null && temp.data != after.data) {
36             temp = temp.next;
37         }
38         if (temp.next != null)
39             temp.next = temp.next.next;
40     }
41
42     // used to insert a node at the start of linked list
43     public void insertLast(int data) {
44         Node current = head;
45         while (current.next != null) {
46             current = current.next; // we'll loop until current.next is null
47         }
48         Node newNode = new Node();
49         newNode.data = data;
50         current.next = newNode;
51     }
52
53     // For printing Linked List
54     public void printLinkedList() {
55         System.out.println("Printing LinkedList (head --> last) ");
56         Node current = head;
57         while (current != null) {
58             current.displayNodeData();
59             current = current.next;
60         }
61         System.out.println();
62     }
63 }
```

## LinkedListMain.java

```
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);  
        // Linked list will be  
        // 2 -> 1 -> 7 -> 6 -> 5  
        Node node=new Node();  
        node.data=1;  
        myLinkedlist.deleteAfter(node);  
        // After deleting node after 1, Linked list will be  
        // 2 -> 1 -> 6 -> 5  
        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, write the java codes to find the middle node

```
1 // find middle element in linkedlist
2 public Node findMiddleNode(Node head) {
3     Node slowPointer, fastPointer;
4     slowPointer = fastPointer = head;
5
6     while(fastPointer !=null) {
7         fastPointer = fastPointer.next;
8         if(fastPointer != null && fastPointer.next != null) {
9             slowPointer = slowPointer.next;
10            fastPointer = fastPointer.next;
11        }
12    }
13
14    return slowPointer;
15 }
```



# Exercise 2: how to reverse a linked list?

```
1 // an iterative solution to Reverse a linked list
2 public static Node reverseLinkedList(Node currentNode) {
3     // For first node, previousNode will be null
4     Node previousNode=null;
5     Node nextNode;
6     while(currentNode!=null) {
7         nextNode=currentNode.next;
8         // reversing the link
9         currentNode.next=previousNode;
10        // moving currentNode and previousNode by 1 node
11        previousNode=currentNode;
12        currentNode=nextNode;
13    }
14    return previousNode;
15 }
```

```
1 // a recursive solution to Reverse a linked list
2 public static Node reverseLinkedList(Node node) {
3     if (node == null || node.next == null) {
4         return node;
5     }
6
7     Node remaining = reverseLinkedList(node.next);
8     node.next.next = node;
9     node.next = null;
10    return remaining;
11 }
```



# Exercise 2: how to reverse a linked list?

```
public static void main(String[] args) {
    LinkedList list = new LinkedList();
    // Creating a linked list
    Node head=new Node(5);
    list.addToTheLast(head);
    list.addToTheLast(new Node(6));
    list.addToTheLast(new Node(7));
    list.addToTheLast(new Node(1));
    list.addToTheLast(new Node(2));

    list.printList(head);
    //Reversing LinkedList
    Node reverseHead=reverseLinkedList(head);
    System.out.println("After reversing");
    list.printList(reverseHead);

}
```

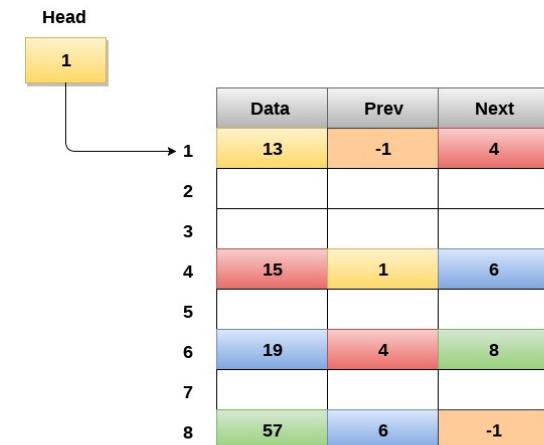
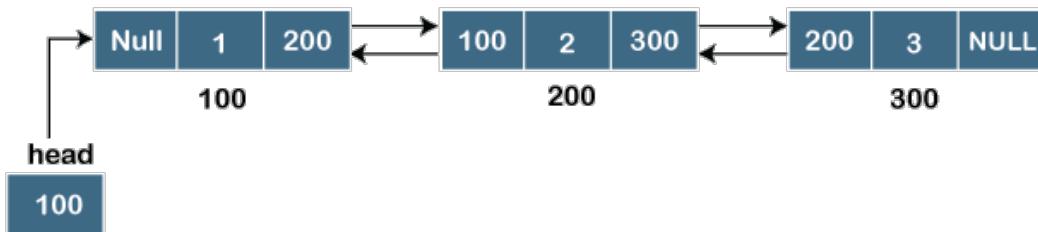
Run above program, you will get following output:

```
5 6 7 1 2
After reversing
2 1 7 6 5
```



# 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 the next node



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

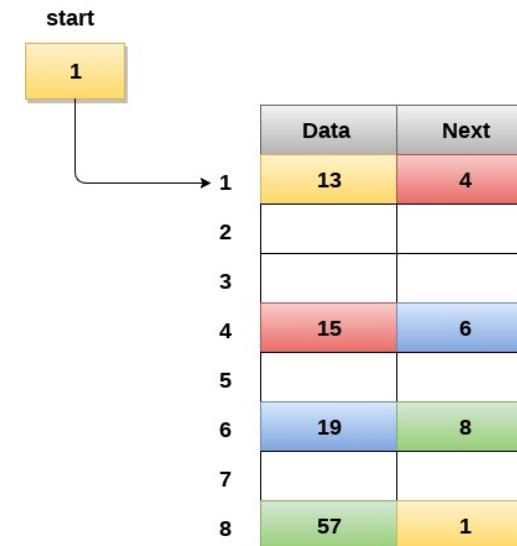
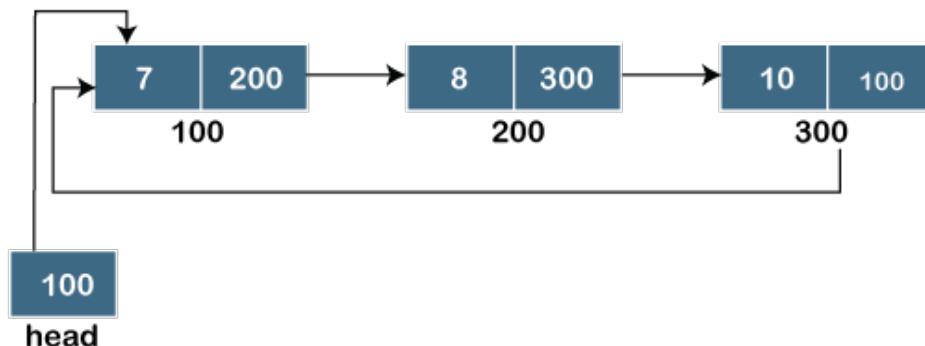
---

- ▶ Doubly linked list is used in navigation systems, for front and back navigation
- ▶ In the browser when we want to use the back or next function to change the tab, it used the concept of a doubly-linked list
- ▶ 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 many operating systems, the thread scheduler maintains a doubly-linked list of all the processes running at any time
  - This makes it 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
  - The last node does not point to any node in a singly linked list
  - The last node links to the first node in a circular linked list
  - The circular linked list has no starting and ending node, so we can traverse in any direction





# 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 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
- ▶ It is also used by the Operating system to share time for different users, generally uses 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
- ▶ In photoshop, word, or any paint, we use this concept in undo function



# Question

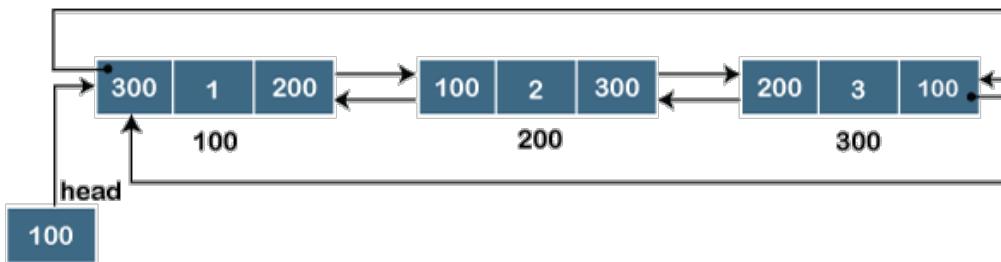
---

- ▶ Both doubly linked list and circular linked list can overcome the limitation of singly linked list. Which one is better?
  - Hint: think the time and space cost



# 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



	Data	Prev	Next
1	A	8	4
2			
3			
4	B	1	6
5			
6	C	4	8
7			
8	D	6	1

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.



# Summary

---

Why insertion is faster in linked list?

- Linked list's each element maintains two pointers (addresses) which points to the both neighbor elements in the list

Does linked list have index?

- It's important to mention that, unlike an array, linked lists do not have built-in indexes
- In order to find a specific point in the linked list, you need to start at the beginning and traverse through each node, one by one, until you find what you're looking for



# Recommended reading

---

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