UNIVERSITI TEKNOLOGI MARA

CSC508 Data Structures

Topic 3 : Linked List

Compiled & edited by: Zahid Zainal

Recap

- Collection hierarchy
- Array definition
- Array implementation
  - `ArrayList`
  - User-defined

Topic Structure

- Linked list
  - Definition
  - Characteristics
  - Properties
- `LinkedList` class
- Linked list operation

Learning Outcomes

- At the end of this lesson, students should be able to:
  - Define the linked list data structure and its characteristics
  - Implement built-in `LinkedList` class
  - Define linked list operations

Linked List Definition

- A list of items, called nodes, in which the order of the nodes is determined by the address, called the link, stored in each node.
  - a series of connected nodes, where each node is a data structure
- Every node in a linked list has two components:
  - one to store relevant information
  - one to store address (the link) of next node

Structure of a node

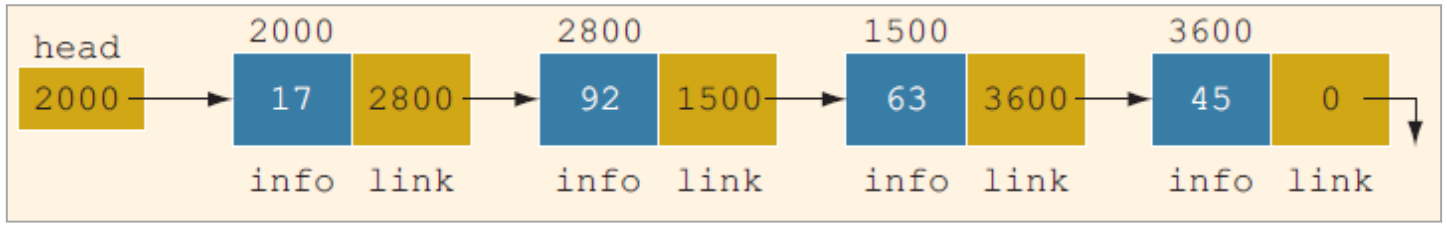
Linked List Characteristics

- Homogeneous
  - Node should be of the same structure.
- Unlimited size
  - The node will be created upon addition. No predefined size.
- Sequential Access
  - Traversing the linked list starting from first/head node.

Linked List Properties

- Address of first node in list stored in separate location, called the head or first
- Data type of each node depends on the specific application — kind of data being processed
- Link component of each node is a reference variable
- Data type of this reference variable is node type itself

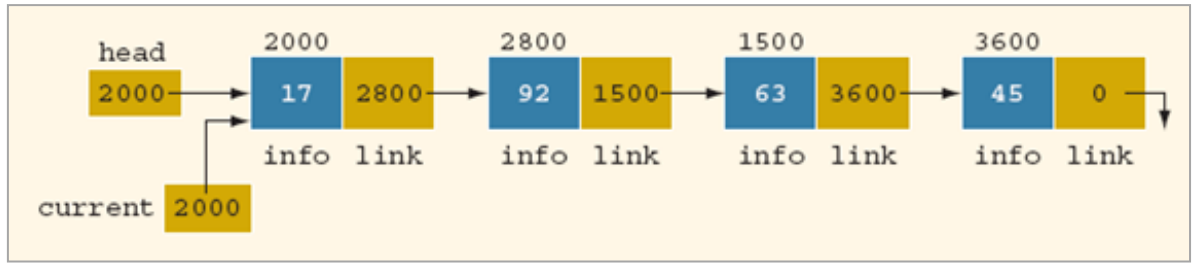
Linked List Properties (cont.)



	Value
head	2000
head.info	17
head.link	2800
head.link.info	92

Linked List Properties (cont.)

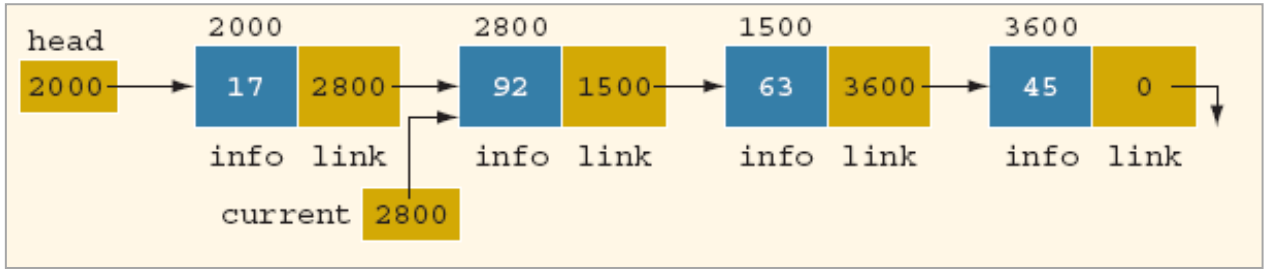
- `NodeType current = head;`
  - Copies value of `head` into `current`



	Value
current	2000
current.info	17
current.link	2800
current.link.info	92

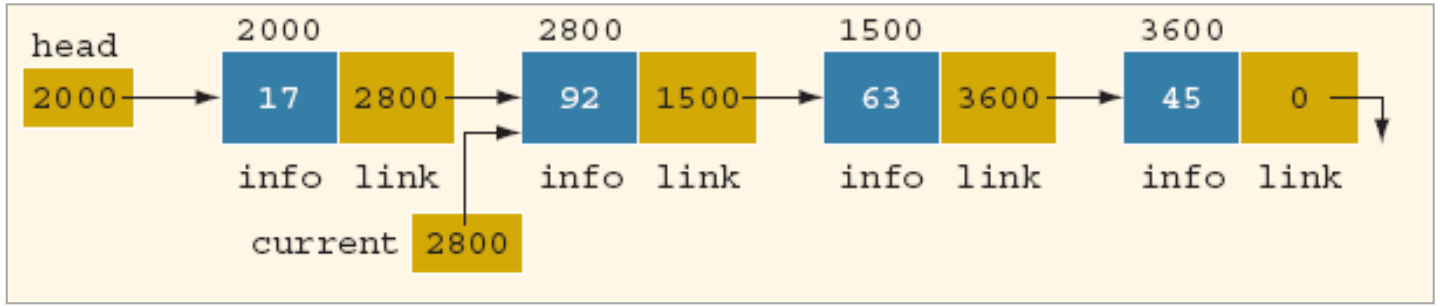
Linked List Properties (cont.)

- `current = current.link;`
  - Copies value of `current.link` (2800) into `current`



	Value
current	2800
current.info	92
current.link	1500
current.link.info	63

Linked List Properties (cont.)



	Value
head.link.link	1500
head.link.link.info	63
head.link.link.link	3600
head.link.link.link.info	45
current.link.link	3600
current.link.link.info	45
current.link.link.link	0 (that is, NULL)
current.link.link.link.info	Does not exist

Runtime error

Linked List Operation

- Create
- Insert element
  - Insert at the beginning, Insert at the end, Insert at specified location
- Remove element
  - Remove the first node, Remove the last node, Remove specified node
- Searching

Built-in `LinkedList` Class

Method	Behavior
`public void add(int index, Object obj)`	Inserts object obj into the list at position index.
`public void addFirst(Object obj)`	Inserts object obj as the first element of the list.
`public void addLast(Object obj)`	Adds object obj to the end of the list.
`public Object get(int index)`	Returns the item at position index.
`public Object getFirst()`	Gets the first element in the list. Throws `NoSuchElementException` if list is empty.
`public Object getLast()`	Gets the last element in the list. Throws `NoSuchElementException` if list is empty.
`public boolean remove(Object obj)`	Removes the first occurrence of object obj from the list. Returns `true` if the list contained object obj; otherwise, returns `false`.
`public int size()`	Returns the number of objects contained in the list.

LinkedList Class Implementation

```
import java.util.Scanner;
import java.util.LinkedList;

public class BuiltInLL {

    public static void main(String[] args) {
        LinkedList<Integer> numList = new LinkedList<Integer>();
        Scanner input = new Scanner(System.in);

        for(int i = 0; i< 3; i++) {
            System.out.println("Enter an integer : ");
            int number = input.nextInt();
            numList.addFirst(number);
        }
        System.out.println("The list size : " + numList.size());
```

Create new LinkedList object

Add new element as first node

```
Enter an integer :
5
Enter an integer :
8
Enter an integer :
6
The list size : 3
```

Get the size of the linked list

LinkedList Class Implementation (cont.)

Accessing the list element

```
for (int i = 0; i < numList.size(); i++)
    System.out.println(numList.get(i));

numList.removeLast();

System.out.println("\nAfter removal");
for (int i = 0; i < numList.size(); i++)
    System.out.println(numList.get(i));
```

Remove the last node

```
6
8
5

After removal
6
8
```

Linked List Traversal

- Process of visiting and accessing nodes in a linked list
- Needed for the following operation
  - Insert at the end of the list, insert at specified location
  - Remove last node, remove node at specified location
  - Printing the link list
  - Check the size
  - Search element

Creating a Linked List

- Define the node structure


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

- Define the linked list class

```
class MyLinkedList{
    Node head;
    int size;

    MyLinkedList(){
        head = null;
        size = 0;
    }
```

Initialize the linked list attribute

Insert node

- Insert at the beginning

```
public void insertFirst(int x) {
    Node newNode = new Node();
    newNode.data = x;
    newNode.next = head;
    head = newNode;
    size++;
}
```

Create and populate the new node

Link head to the new node

Update the size

Insert node (cont.)

- Insert at the end

If the list is empty, always insert as first element

Create and populate the new node

Traverse until the last node

```
public void insertLast (int x) {
    if (head == null)
        insertFirst(x);
    else {
        Node newNode = new Node();
        newNode.data = x;
        newNode.next = null;
        Node temp = head;

    while (temp.next != null)
        temp = temp.next;
        temp.next = newNode;
        size++;
    }
}
```

Reference variable for traversal

Update the size

Printing linked list

```
public void printLL() {
    if (head == null)
        System.out.println("List is empty");
    else {
        Node temp = head;
        while (temp != null) {
            System.out.println(temp.data);
            temp = temp.next;
        }
    }
}
```

Check whether list is empty

Reference variable for traversal

Traverse until the end of the list

Print data on every node visit

Testing - 1

```
public class Main {

    public static void main(String[] args) {
        MyLinkedList LL1 = new MyLinkedList();
        LL1.printLL();
        System.out.println("***");
        LL1.insertFirst(6);
        LL1.insertFirst(8);
        LL1.insertLast(3);
        LL1.insertLast(5);
        LL1.printLL();
    }
//continue next
```

```
List is empty
***
8
6
3
5
```

Remove node

- Remove the first node

```
public void removeFirst() {
    if (head == null)
        System.out.println("Empty list");
    else {
        head = head.next;
        size--;
    }
}
```

Check whether list is empty

Update the size

Break the link to the first node and link to next node

Remove node (cont.)

- Remove the last node

Check whether list is empty

Case where there is only 1 node in the list

Traverse until the second last node

Break the link to the last node

```
public void removeLast() {
    if (head == null)
        System.out.println("Empty list");
    else if (size == 1){
        removeFirst();
        size--;
    }
    else {
        Node temp = head;
        while (temp.next.next != null)
            temp = temp.next;
        temp.next = null;
        size--;
    }
}
```

Testing - 2

```
//from previous
        LL1.removeFirst();
        LL1.removeLast();
        LL1.printLL();
        System.out.println("***");
        LL1.removeLast();
        LL1.removeLast();
        LL1.printLL();
    }
}
```

```
6
3
***
List is empty
```

Summary

- A list of items, called nodes, in which the order of the nodes is determined by the address, called the link, stored in each node.
- Linked list characteristics – Homogeneous, Unlimited side, Sequential access.
- `LinkedList` class implementation
- Linked list operation – Create, Insert, Remove, Print

Next Topic…

- Linked list variation
  - Doubly linked list
  - Circular linked list

References

- Carrano, F. & Savitch, W. 2005. *Data Structures and Abstractions with Java, 2nd ed. Prentice-Hall.*
- Malik D.S, & Nair P.S., Data Structures Using Java, Thomson Course Technology, 2003.
- Rada Mihalcea, CSCE 3110 Data Structures and Algorithm Analysis notes, U of North Texas.