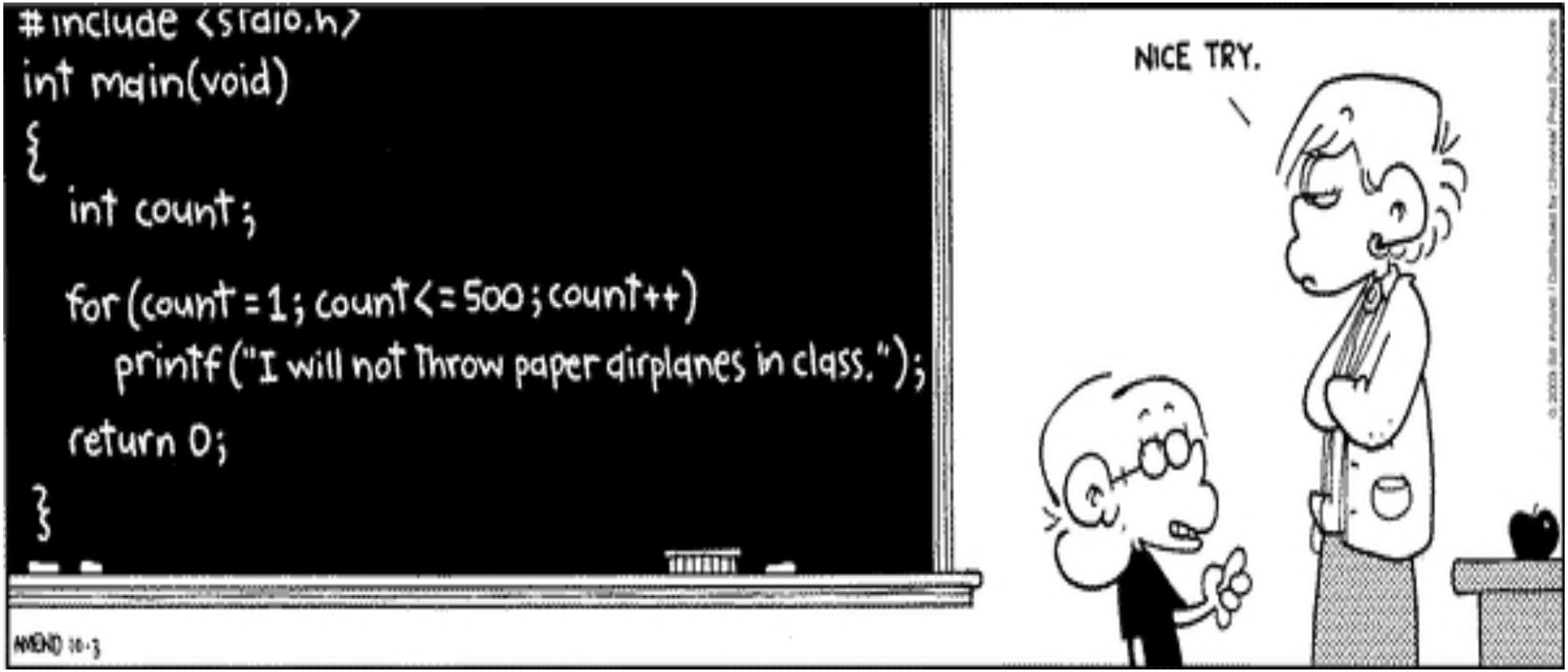
CUS 1126: Introduction to Data Structures

Lecture : Linked lists

Instructor: Nikhil Yadav


#include <stdio.h>
int main(void)
{
int count;
for (count = 1; count <= 500; count++)
printf("I will not throw paper airplanes in class.");
return 0;
}
NICE TRY.

Motivation

- Suppose we have an array: 1,4,10,19,6
- We want to insert a 7 between the 4 and the 10
- What do we need to do?

Linked Lists
aka “Reference-Based Lists”

- Linked list
  - List of items, called nodes
  - The order of the nodes is determined by the address, called the link, stored in each node

Linked Lists
aka "Reference-Based Lists"

- Linked list
  - List of items, called nodes
  - The order of the nodes is determined by the address, called the link, stored in each node

- Every node (except the last node) contains the address of the next node

Linked Lists
aka "Reference-Based Lists"

- Linked list
  - List of items, called nodes
  - The order of the nodes is determined by the address, called the link, stored in each node
- Every node (except the last node) contains the address of the next node
- Components of a node
  - Data: stores the relevant information
  - Link: stores the address of the next node

Linked Lists

Structure of a node:

- Head or first
  - Holds the address of the first node in the list

- The info part of the node can be either a value of a primitive type or a reference to an object

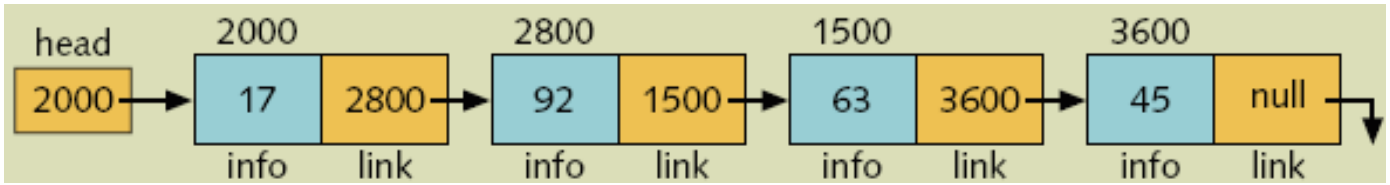
Linked Lists (continued)

- Class Node
  - Represents nodes on a list
  - It has two instance variables
    - `info` (of type `int`, but it can be any other type)
    - `link` (of type `Node`)

```
public class Node {
    public int info;
    public Node link;
}
```

Linked List: Some Properties

Linked list with four nodes



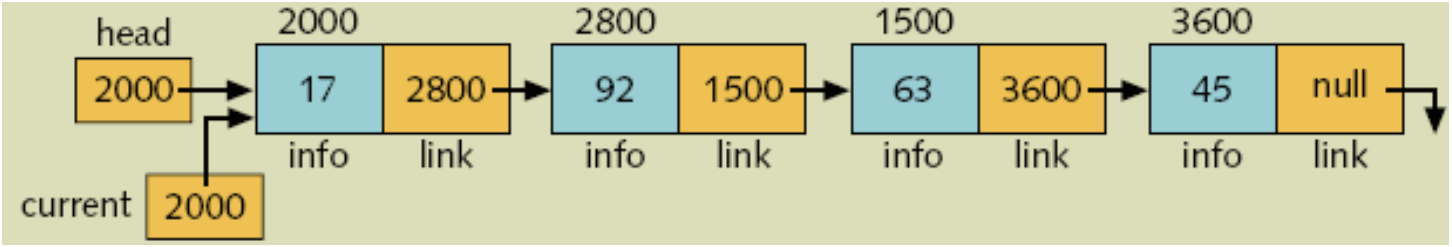
Values of head and some of the nodes in the linked list above

	Value	Explanation
`head`	`2000`	
`head.info`	`17`	Because `head` is `2000` and the `info` of the node at location `2000` is `17`
`head.link`	`2800`	
`head.link.info`	`92`	Because `head.link` is `2800` and the `info` of the node at location `2800` is `92`

Linked List: Some Properties

- Now consider the statement

```
current= head;
```



Linked list after `current = head;` executes

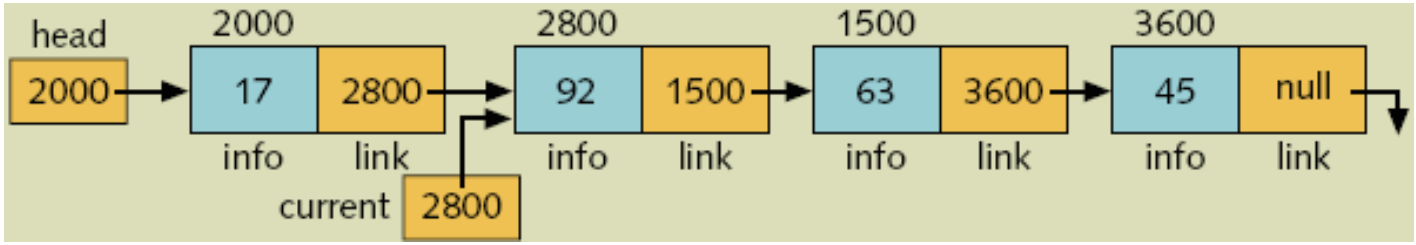
Values of `current` and some of the nodes of the linked list above

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

Linked List: Some Properties

- Now consider the statement

```
current = current.link;
```

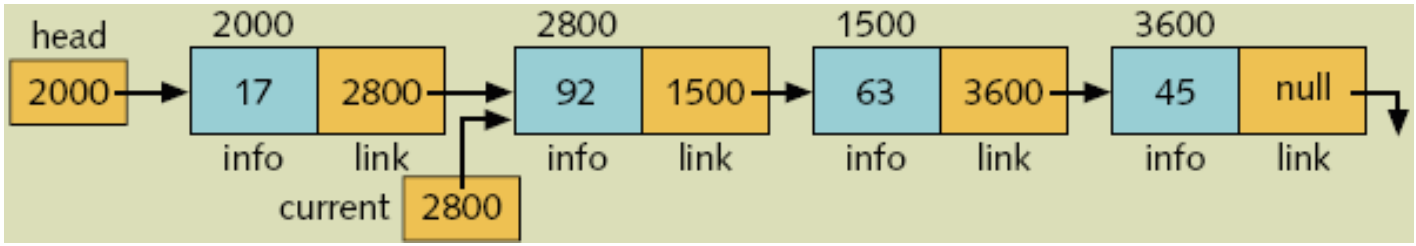


List after the statement `current = current.link;`

Values of `current` and some of the nodes of the linked list above

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

Linked List: Some Properties



Values of various reference variables and nodes in linked list above

	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`	`null`
`current.link.link.link.info`	Does not exist

Traversing a Linked List

- Basic operations of a linked list that require the link to be traversed
  - Search the list for an item
  - Insert an item in the list
  - Delete an item from the list

Traversing a Linked List

- Basic operations of a linked list that require the link to be traversed
  - Search the list for an item
  - Insert an item in the list
  - Delete an item from the list

- You cannot use `head` to traverse the list
  - You would lose the nodes of the list
  - Use another reference variable of the same type as
    - `head`: `current`

Traversing a Linked List

- The following code traverses the list

```
current = head;
while (current != null) {
      //Process current
      current = current.link;
}
```

Exercise

- Write code to print out the data stored in each node in a linked list

Exercise

- Write code to print out the data stored in each node in a linked list

```
current = head;
while (current != null)
{
    System.out.println(current.info + " ");
    current = current.link;
}
```

Exercise

- Write code to set the data in the 5th node to be 10

Exercise

- Write code to set the data in the 5th node to be 10

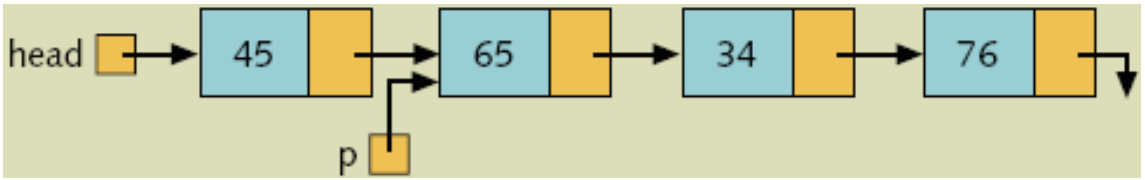
```
current = head;

cnt = 0;

while (cnt < 4 && current != null)
{
    current = current.link;
    cnt++;
}
if (current != null)
{
    current.info = 10;
```

Insertion

- Consider the following linked list:



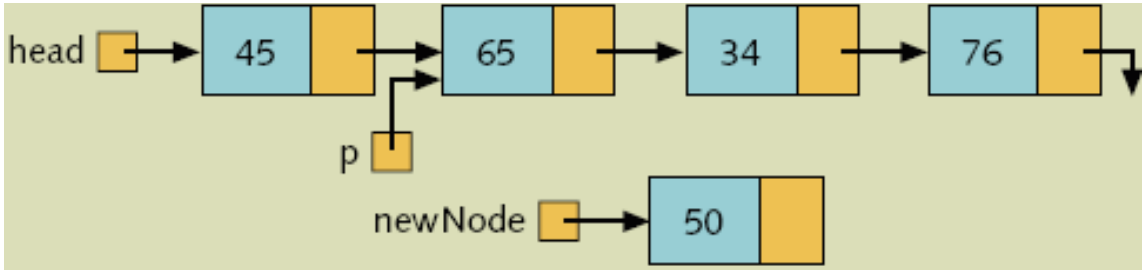
Linked list before item insertion

- You want to create a new node with `info 50` and insert it after `p`

Insertion

- The following statements create and store `50` in the `info` field of a new node

```
Node newNode = new Node();     //create newNode
newNode.info = 50;             //store 50 in the new  node
```



Create `newNode` and store `50` in it

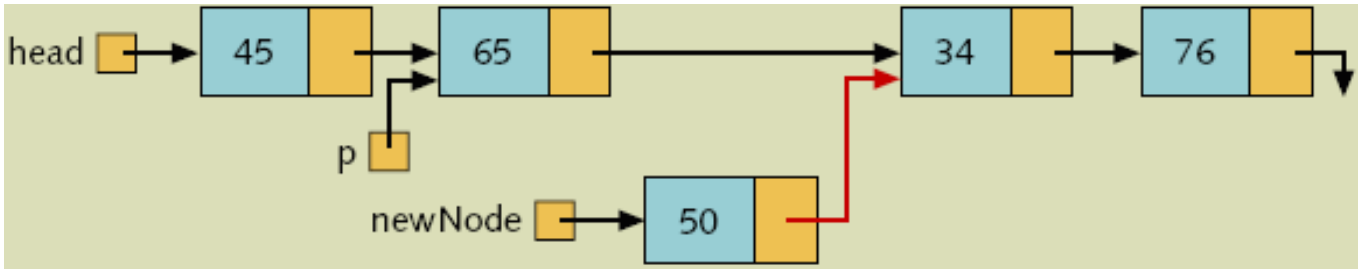
Insertion

- The following statements insert the node in the linked list at the required place

```
newNode.link = p.link;
p.link = newNode;
```

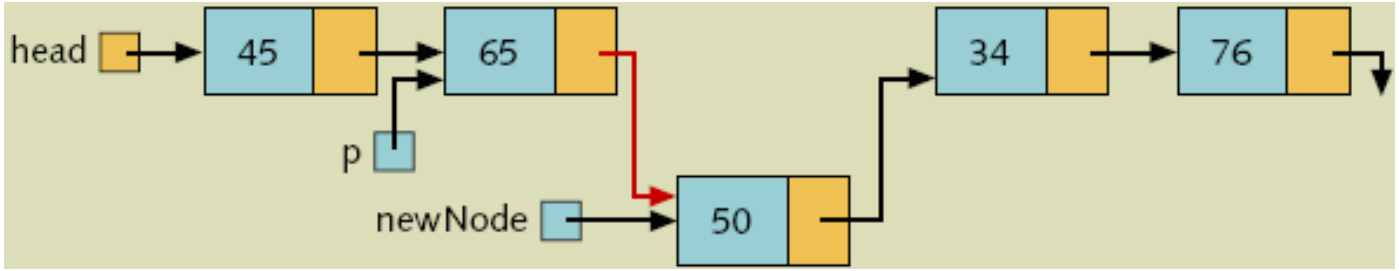
- The sequence of statements to insert the node is very important

Insertion (Continued)



List after the statement

`newNode.link = p.link;` executes

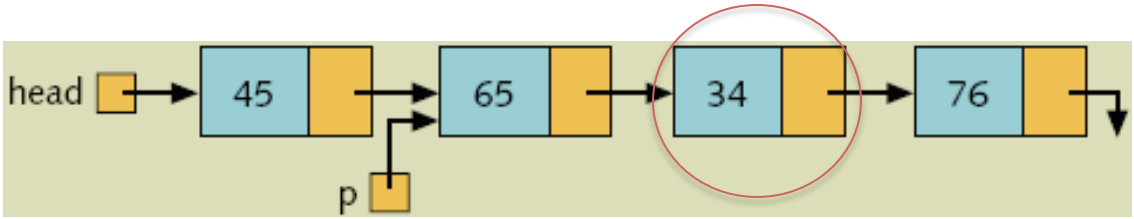


List after the statement

`p.link = newNode;` executes

Deletion

- Consider the following linked list



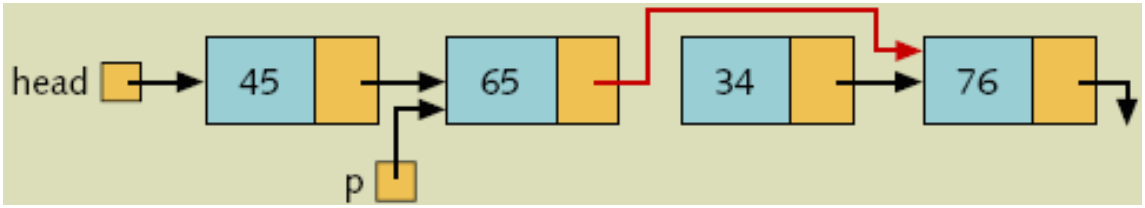
Node to be deleted is with `info 34`

- You want to delete node with `info 34`

Deletion (continued)

- The following statement removes the nodes from the list

```
p.link = p.link.link
```



List after the statement

`p.link = p.link.link;` executes

Deletion (continued)

- Previous statement removed the node
  - However, the memory may still be occupied by this node
- System's automatic garbage collector reclaims memory occupied by unreferenced nodes
  - Use `System.gc();` to run the garbage collector

Building a linked list

- You can build a list in two ways: forward or backward

Building a linked list

- You can build a list in two ways: forward or backward
- Forward manner
  - A new node is always inserted at the end of the linked list

Building a linked list

- You can build a list in two ways: forward or backward
- Forward manner
  - A new node is always inserted at the end of the linked list
- Backward manner
  - A new node is always inserted at the beginning of the linked list

Building a linked list (forward)

- You need three reference variables

Building a linked list (forward)

- You need three reference variables
  - One to point to the front of the list
    - Cannot be moved

Building a linked list (forward)

- You need three reference variables
  - One to point to the front of the list
    - Cannot be moved
  - One to point to the last node of the list

Building a linked list (forward)

- You need three reference variables
  - One to point to the front of the list
    - Cannot be moved
  - One to point to the last node of the list
  - One to create the new node

Building a linked list (forward)

- You need three reference variables
  - One to point to the front of the list
    - Cannot be moved
  - One to point to the last node of the list
  - One to create the new node

- Next two slides show the code for creating a linked list forward

Building a linked list (forward)

```
Node buildListForward()
{
   Node first, newNode, last;
   int num;
   System.out.println("Enter integers ending with
   -999:");
   num = console.nextInt();
   first = null;
   while (num != -999)
   {
      newNode = new Node();
      newNode.info = num;
      newNode.link = null;
```

Building a linked list (forward… continued)

```
    if (first == null)
       {
          first = newNode;
          last = newNode;
       }
    else
       {
          last.link = newNode;
          last = newNode;
       }
       num = console.nextInt();
    }//end while
    return first;
}//end buildListForward
```

Building a linked list (backward)

- You only need two reference variables

Exercise

- Write code to take in an array of ints and returns the head reference to a linked list of the ints

```
public Node createLinkedList(int[] a) {




}
```

- Write code to take in an array of ints and returns the head reference to a linked list of the ints

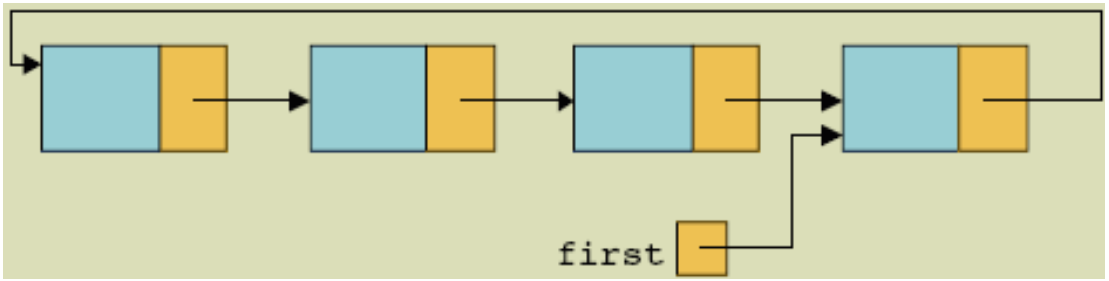
```
public Node createLinkedList(int[] a) {
  Node head = new Node();
  head.info = a[length-1];
  head.link = null;

  for (int i = a.length-2; i >=0; i--) {
    Node n = new Node();
    n.info = a[i];
    n.link = head.link;
    head = n;
  }
  return head;
}
```

Circular Linked Lists

- A linked list in which the last node points to the first node

- It is convenient to make `first` point to the last node



Circular linked list with more than one node

Thank you