Linked Lists

Computer Programming II
Profs. H. Harmanani and W. Keirouz
Lebanese American University
Byblos

Linked Lists

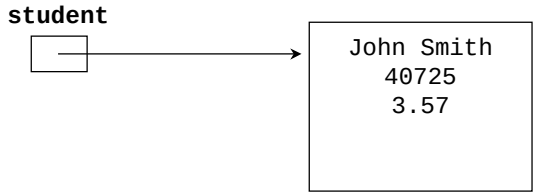
- **Chapter 4 focuses on:**
  - Dynamic structures
  - Abstract Data Types (ADTs)
  - Linked lists
  - Most common operations on linked lists.
- **Chapter 5 introduces the often-used data public classure of linked lists.**

Static vs. Dynamic Structures

- **A *static* data structure has a fixed size**
- **This meaning is different than those associated with the `static` modifier**
- **Arrays are static; once you define the number of elements it can hold, it doesn't change**
- **A *dynamic* data structure grows and shrinks as required by the information it contains**

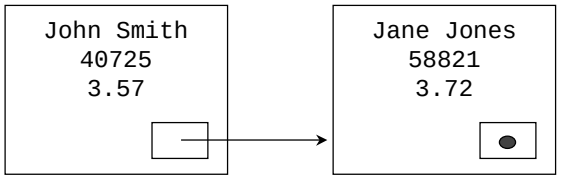
Object References

- **An *object reference* is a variable that stores the address of an object**
- **A reference can also be called a *pointer***
- **They are often depicted graphically:**



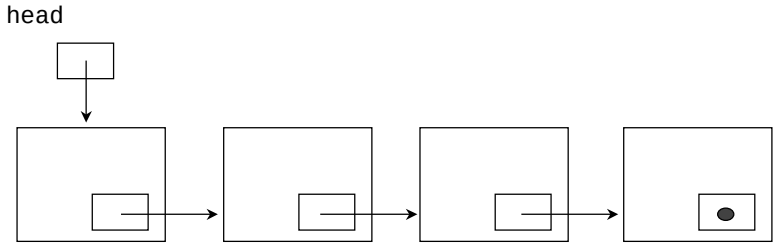
References as Links

- **Object references can be used to create *links* between objects**
- **Suppose a `Student` class contained a reference to another `Student` object**



Linked Lists

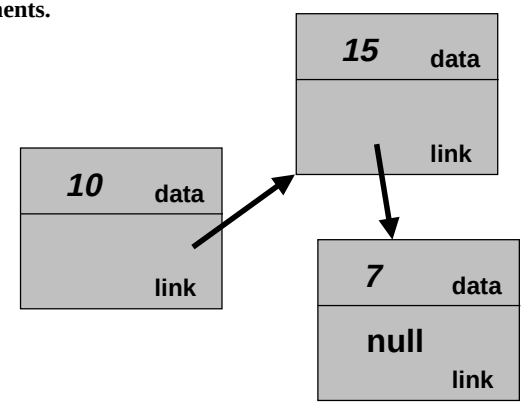
- **A linked list is a sequence of elements arranged one after another, with each element connected to the next element by a *link***
- **Need a special reference that points to the list's first node, called the *head***
  - Sometimes we may need a reference to the list's last node



Declarations for Linked Lists

- **For this presentation, each node in the linked list is a class, as shown.**
- **The actual class has methods for getting and setting the two instance variables. There is also a constructor that creates a new IntNode with specified data and link components.**

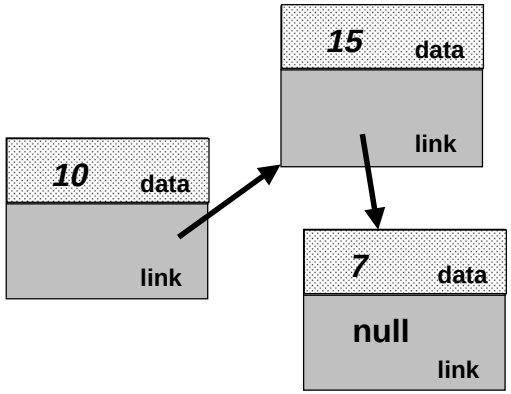
```
public class IntNode
{
    private int data;
    private IntNode link;
    …
}
```



Declarations for Linked Lists

- **The data portion of each node is an int.**
  - How would we create a linked list of real numbers?

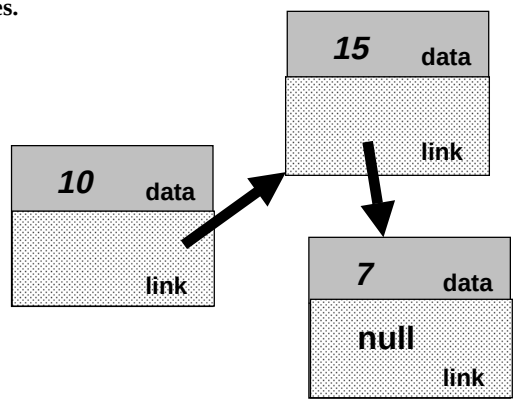
```
public class IntNode
{
    private int data;
    private IntNode link;
    …
}
```



Declarations for Linked Lists

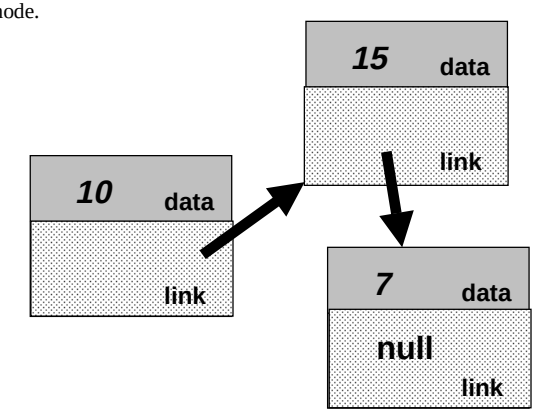
- **Each IntNode also contains a link which refers to another IntNode.**
- **Also inside each IntNode is a second member variable called link. The purpose of the link member variable is to contain a reference to the next IntNode in the sequence of nodes.**

```
public class IntNode
{
    private int data;
    private IntNode link;
    ...
}
```



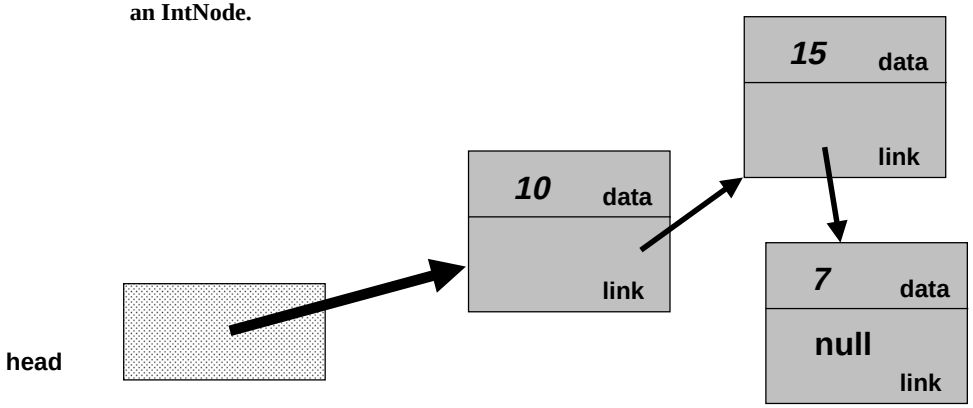
Null Reference

- **The link link field of the final node in a linked list is a special reference value called null**
  - Null means "This variable doesn't refer to anything." This makes sense because there is no node after the final node.



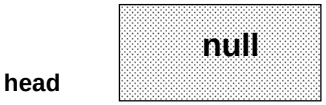
Declarations for Linked Lists

- **A program can keep track of the front node by using a variable such as** head **in this example.**
- **Notice that head is not an IntNode -- it is a reference to an IntNode.**



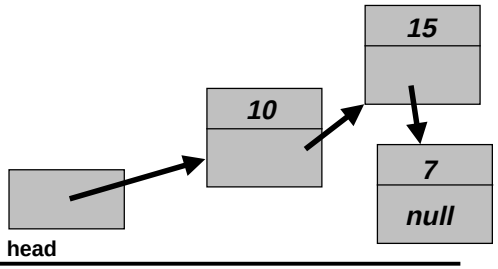
Declarations for Linked Lists

- **A program can keep track of the front node by using an IntNode reference variable such as** head**.**
- **Notice that head is not an IntNode -- it is a reference to an IntNode.**
- **We represent the empty list by storing** null **in the head reference.**



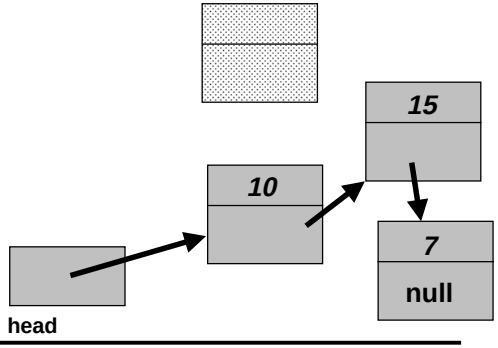
Inserting an IntNode at the Front

We want to add a new entry, 13, to the front of the linked list shown here.



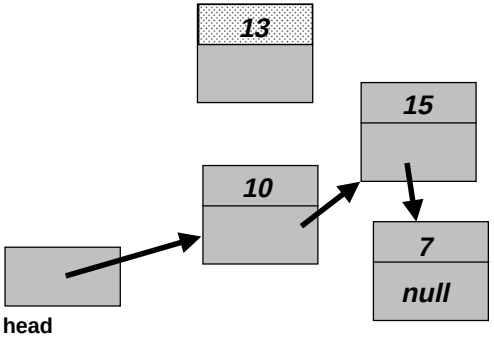
Inserting an IntNode at the Front

❶ **Create a new node...**



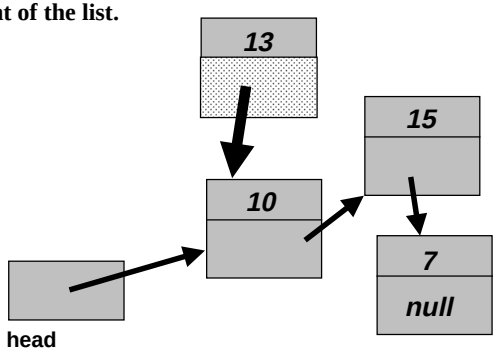
Inserting an IntNode at the Front

❶ **Create a new node...**
❷ **Place the data in the new node's data field.**



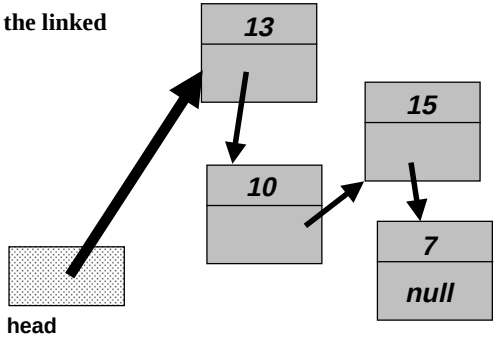
Inserting an IntNode at the Front

❶ **Create a new node...**
❷ **Place the data in the new node's data field....**
❸ **Connect the new node to the front of the list.**



Inserting an IntNode at the Front

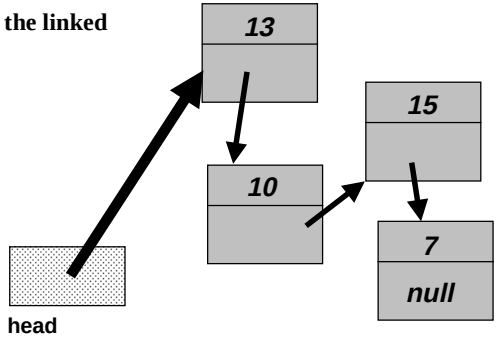
❶ **Create a new node...**
❷ **Place the data in the new node's data field....**
❸ **Connect the new node to the front of the list.**
❹ **Make the head refer to the new head of the linked list.**



Inserting an IntNode at the Front

head = new IntNode(13, head);

❶ **Create a new node...**
❷ **Place the data in the new node's data field....**
❸ **Connect the new node to the front of the list.**
❹ **Make the head refer to the new head of the linked list.**



Linked List – Specification

- **Class Name**
  - Implemented as class `IntNode`.
- **Constructor**
  - `public IntNode(int initialData, intNode initialLink)`
    - Initialize a node with a specified initial data and link to the next node.
    - If the list is empty then the `initialLink` points to null.
    - **Postcondition –** New list contains the specified data and link to the next node

Linked List – Constructor Implementation

```
public IntNode(int initialData, IntNode initialLink)
{
    data = initialData;
    link = initialLink;
}
```

Linked List – Specification

- **Class Name**
  - Implemented as class `IntNode`.
- **Accessor Methods**
  - **`public`** `int getData()`
    - Get the data from this node
    - **Return Value:** Data from this node
  - **`public`** `int getLink()`
    - Get a reference to the node after this node
    - **Return Value:** a reference to the node after this node or **null** reference if there is nothing after this node.

Linked List – Specification

- **Modification Methods**
  - **`public`** `int setData(int newdata)`
    - Set the data in this node to `newdata`
    - **Parameters**
      - `newData` – The new data to place in this node
    - **PostCondition**
      - The data of this node has been set to `newData`
    - **Return Value:** Data from this node
  - **`public`** `int setLink(IntNode newLink)`
    - **Parameters**
      - `newLink` – A reference to the node that should appear after this node in the linked list or **null** reference if there should be no node after this node
    - **PostCondition**
      - Link to the node after this node has been set to `newLink.` Any aother node that used to be in this link is no longer connected to this node.

Inserting an IntNode at the Front

```
public IntNode(int initialData, IntNode initialLink)
{
   data = initialData;
   link = initialLink;
}
```

Does the constructor work correctly for the first node on a new list ?

Inserting an IntNode at the Front

```
public IntNode(int initialData, IntNode initialLink)
{
   data = initialData;
   link = initialLink;
}
```

Suppose head is null and we execute the assignment shown here:

```
head = new IntNode(13, head);
```

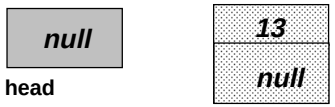
null

head

Inserting an IntNode at the Front

```
public IntNode(int initialData, IntNode initialLink)
{
   data = initialData;
   link = initialLink;
}
```

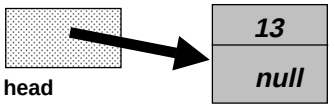
head = new IntNode(13, head);



Inserting an IntNode at the Front

```
public IntNode(int initialData, IntNode initialLink)
{
   data = initialData;
   link = initialLink;
}
```

head = new IntNode(13, head);

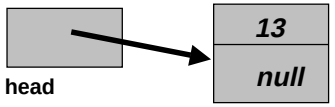


Inserting an IntNode at the Front

```
public IntNode(int initialData, IntNode initialLink)
{
   data = initialData;
   link = initialLink;
}
```

When the statement finishes, the linked list has one node, containing 13.

head = new IntNode(13, head);



Caution!

- **Always make sure that your linked list methods work correctly with an empty list.**

Pseudocode for Inserting IntNodes

- **IntNodes are often inserted at places other than the front of a linked list.**
- **There is a general pseudocode that you can follow for any insertion function. . .**

Pseudocode for Inserting IntNodes

❶ **Determine whether the new node will be the first node in the linked list. If so, then there is only one step:**

```
head = new IntNode(newEntry, head);
```


Pseudocode for Inserting IntNodes

❷ Otherwise (if the new node will not be first):

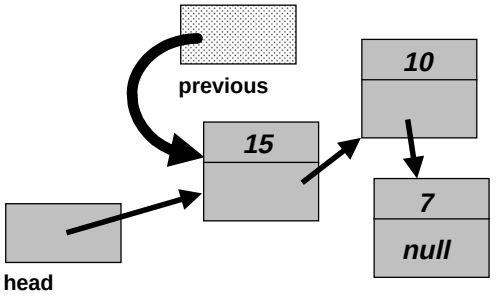
❒ Start by setting a reference named **previous** to refer to the node which is just before the new node's position.

Pseudocode for Inserting IntNodes

❷ Otherwise (if the new node will not be first):

❒ Start by setting a reference named **previous** to refer to the node which is just before the new node's position.

In this example, the new node will be the second node

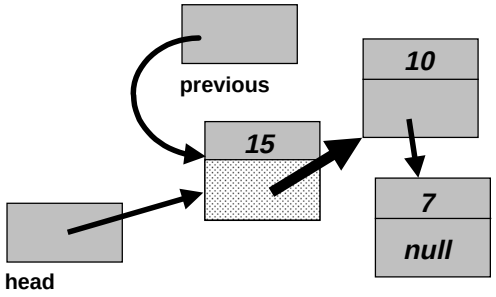


Pseudocode for Inserting IntNodes

- Otherwise (if the new node will not be first):
  - Start by setting a reference named previous to refer to the node which is just before the new node's position

Look at the link which is **in the node** previous

What is the name of this link?

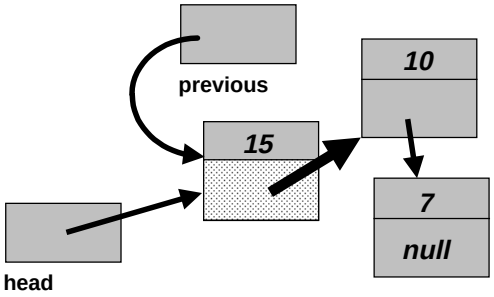


Pseudocode for Inserting IntNodes

- Otherwise (if the new node will not be first):
  - Start by setting a reference named previous to refer to the node which is just before the new node's position

This link is called **previous.link**

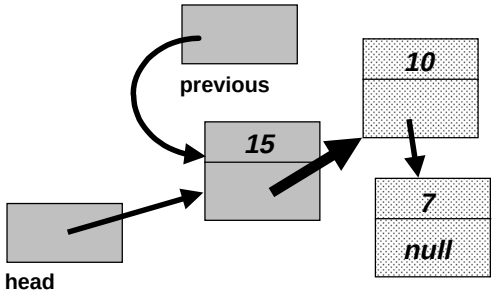
What is the name of this link ?



Pseudocode for Inserting IntNodes

- Otherwise (if the new node will not be first):
  - Start by setting a reference named previous to refer to the node which is just before the new node's position

previous.link refers to the head of a small linked list, with 10 and 7

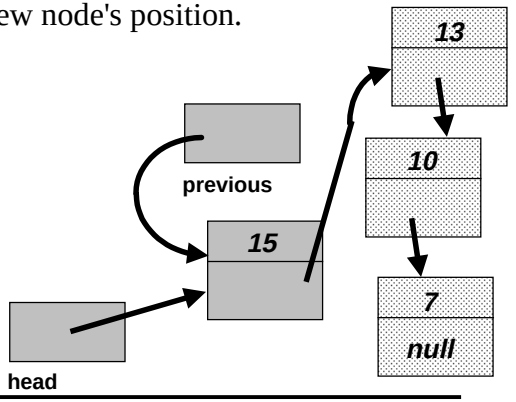


Pseudocode for Inserting IntNodes

- Otherwise (if the new node will not be first):
  - Start by setting a reference named previous to refer to the node which is just before the new node's position.

The new node must be inserted at the front of this small linked list.

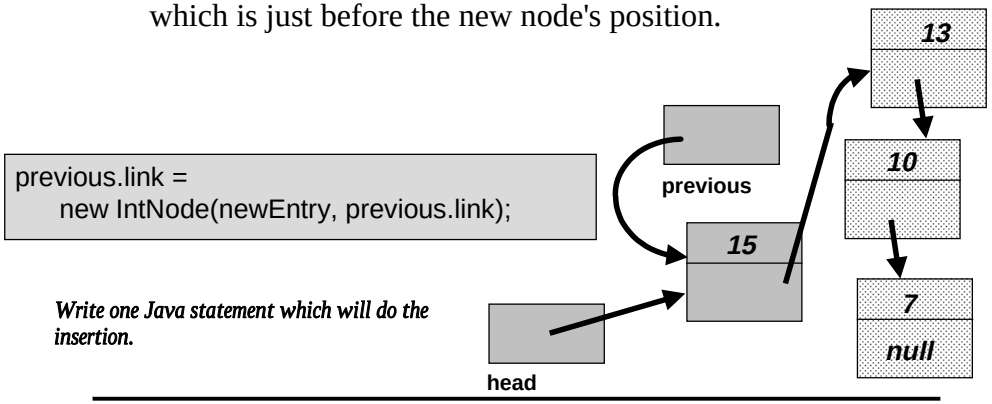
Write one Java statement which will do the insertion.



Pseudocode for Inserting IntNodes

❷ Otherwise (if the new node will not be first):

❒ Start by setting a reference named previous to refer to the node which is just before the new node's position.



Pseudocode for Inserting IntNodes

❶ **Determine whether the new node will be the first node in the linked list. If so, then there is only one step:**

```
head = new IntNode(newEntry, head);
```

❷ Otherwise (if the new node will not be first):

❒ Set a reference named previous to refer to the node which is just before the new node's position.

❒ Execute the step:

```
previous.link =
    new IntNode(newEntry, previous.link);
```


Pseudocode for Inserting IntNodes

- **The process of adding a new node in the middle of a list can also be incorporated as a separate method. This function is called addNodeAfter in the linked list toolkit of Section 4.2.**

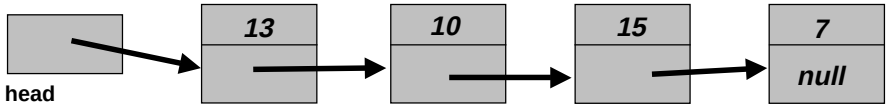
Pseudocode for Removing IntNodes

- **IntNodes often need to be removed from a linked list.**
- **As with insertion, there is a technique for removing a node from the front of a list, and a technique for removing a node from elsewhere.**
- **We'll look at the technique for removing a node from the front of a linked list.**

Removing the Head IntNode

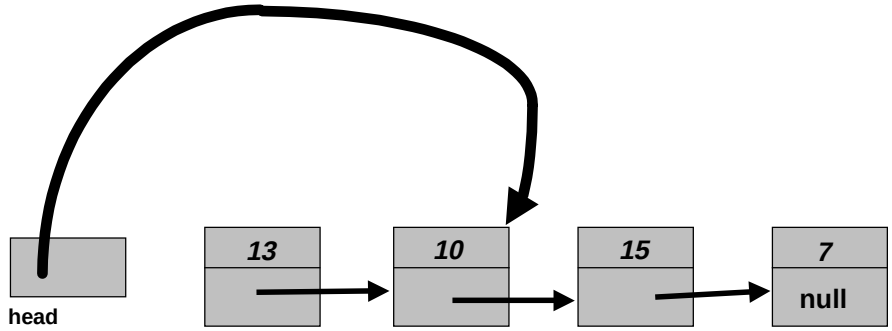
head = head.link;

Draw the change that this statement will make to the linked list.



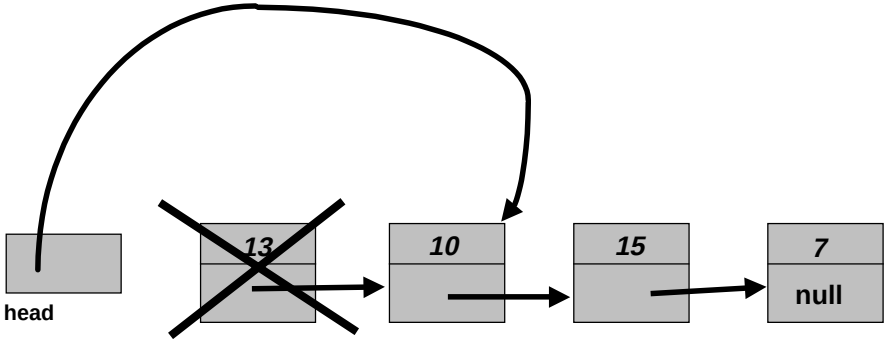
Removing the Head IntNode

head = head.link;



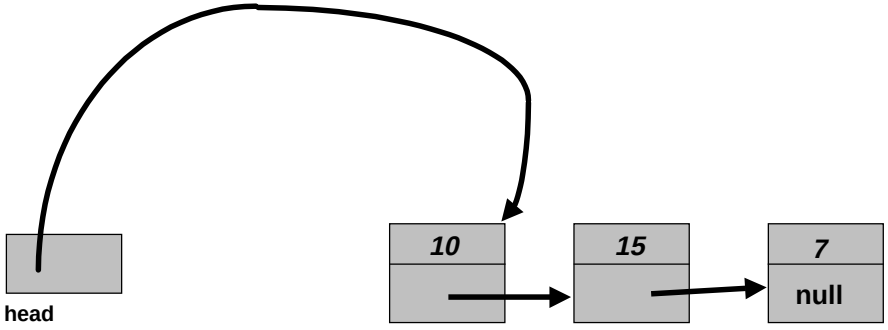
Removing the Head IntNode

head = head.link;



Removing the Head IntNode

Here's what the linked list looks like after the removal finishes.



THE END