AUBURN
UNIVERSITY

SAMUEL GINN
COLLEGE OF ENGINEERING

Linked Structures

COMP 2210 – Dr. Hendrix

A Bag collection

Revisit the Bag collection with a look at an alternate implementation that uses dynamic memory for the physical storage instead of an array.

```
public interface Bag<T> {
   boolean     add(T element);
   boolean     remove(T element);
   boolean     contains(T element);
   int         size();
   boolean     isEmpty();
   Iterator<T> iterator();
}
```

A Bag collection

Revisit the Bag collection with a look at an alternate implementation that uses dynamic memory for the physical storage instead of an array.

```
public interface Bag<T> {
   boolean      add(T element);
   boolean      remove(T element);
   boolean      contains(T element);
   int          size();
   boolean      isEmpty();
   Iterator<T> iterator();
}
```

```
public class ArrayBag<T> implements Bag<T> {

   private T[] elements;
   . . .
}
```

A Bag collection

Revisit the Bag collection with a look at an alternate implementation that uses dynamic memory for the physical storage instead of an array.

```
public interface Bag<T> {
   boolean      add(T element);
   boolean      remove(T element);
   boolean      contains(T element);
   int          size();
   boolean      isEmpty();
   Iterator<T> iterator();
}
```

```
public class ArrayBag<T> implements Bag<T> {

   private T[] elements;
   . . .
}
```

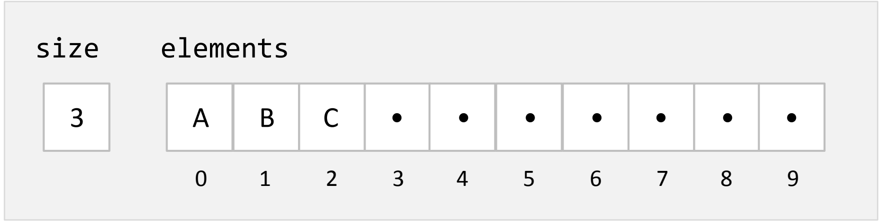
```
public class LinkedBag<T> implements Bag<T> {

   private ???;
   . . .
}
```

ArrayBag

```
public class ArrayBag<T> implements Bag<T> {

   private T[] elements;
   private int size;
   . . .
}
```

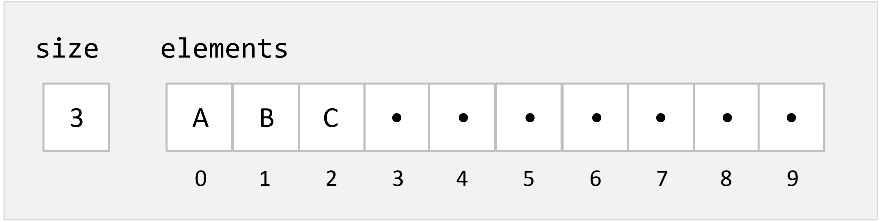
ArrayBag

```
public class ArrayBag<T> implements Bag<T> {

   private T[] elements;
   private int size;
   . . .
}
```



ArrayBag

```
public class ArrayBag<T> implements Bag<T> {

   private T[] elements;
   private int size;
   . . .
}
```

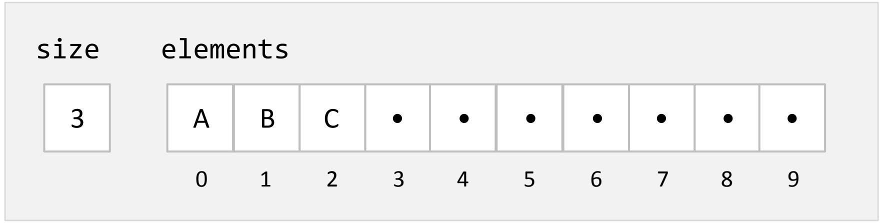


Advantages of using an array:

- fast random access to any element
- efficient use of memory
- built into the language; a “common currency” for any data storage scheme

ArrayBag

```
public class ArrayBag<T> implements Bag<T> {

   private T[] elements;
   private int size;
   . . .
}
```



Advantages of using an array:

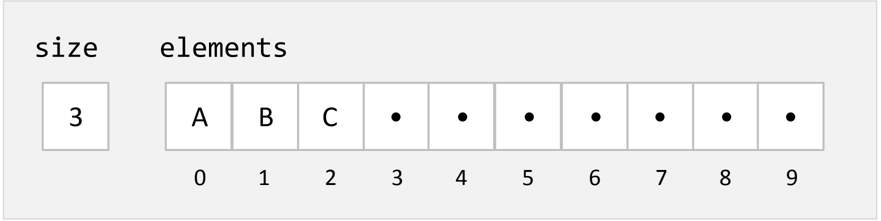
- fast random access to any element
- efficient use of memory
- built into the language; a "common currency" for any data storage scheme

Disadvantages of using an array:

- inefficient to insert or delete anywhere but the end; must shift left/right
- need to "resize" when full/sparse

ArrayBag

```
public class ArrayBag<T> implements Bag<T> {

   private T[] elements;
   private int size;
   . . .
}
```



Advantages of using an array:

- fast random access to any element
- efficient use of memory
- built into the language; a "common currency" for any data storage scheme

Disadvantages of using an array:

- inefficient to insert or delete anywhere but the end; must shift left/right
- need to "resize" when full/sparse

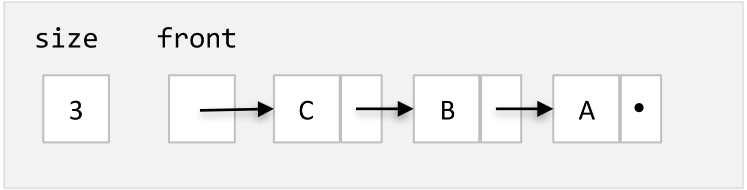
A storage scheme using dynamic memory will address these disadvantages at the cost of losing random access.

LinkedBag

```java
public class LinkedBag<T> implements Bag<T> {

    private ??? front;
    private int size;
    . . .
}
```

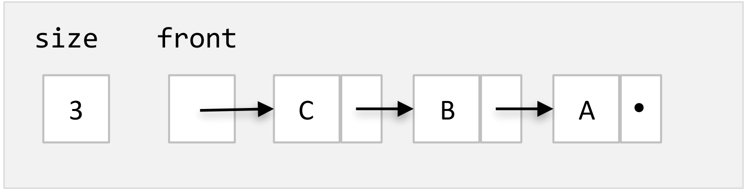
LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private ??? front;
   private int size;
   . . .
}
```



LinkedBag

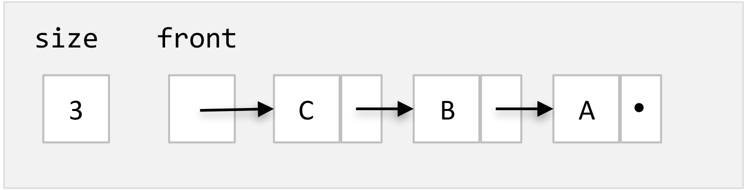
```
public class LinkedBag<T> implements Bag<T> {

   private ??? front;
   private int size;
   . . .
}
```



Individual containers are explicitly linked together. Each container holds one element and a reference to another container.

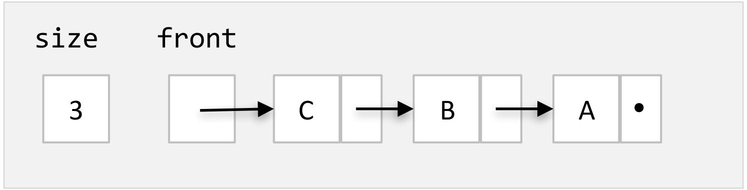
LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .






}
```



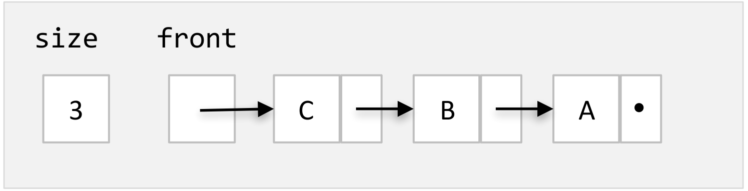
LinkedBag

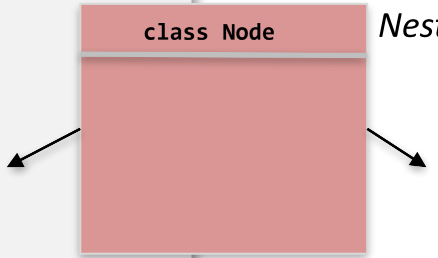
```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {



      . . .

   }

}
```



LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {



      . . .

   }

}
```

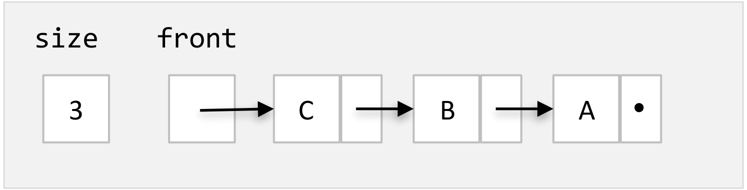




Nested or top-level?

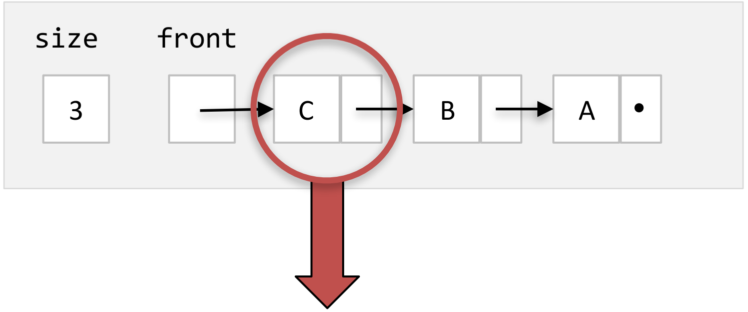
LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {



      . . .
   }


}
```



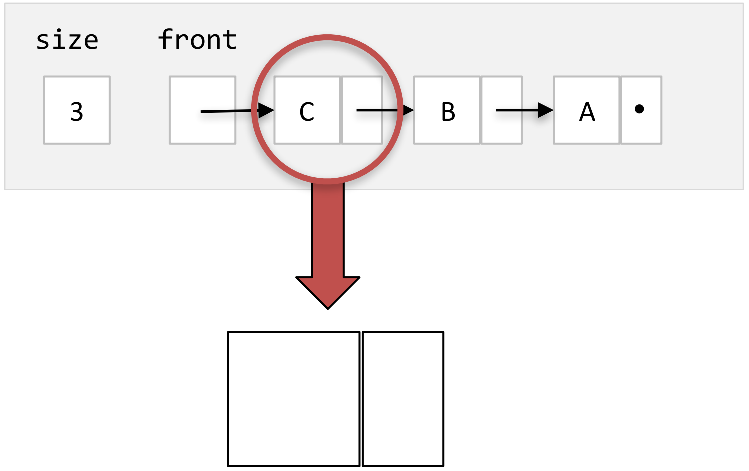
LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {



      . . .
   }

}
```



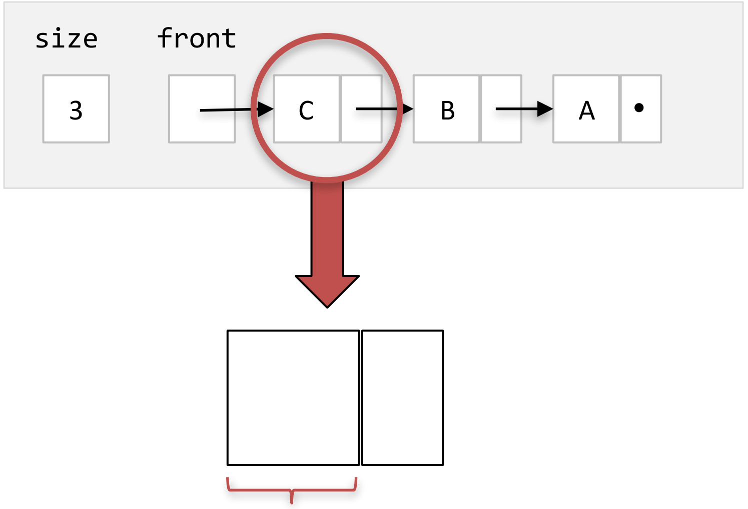
LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {



      . . .
   }


}
```



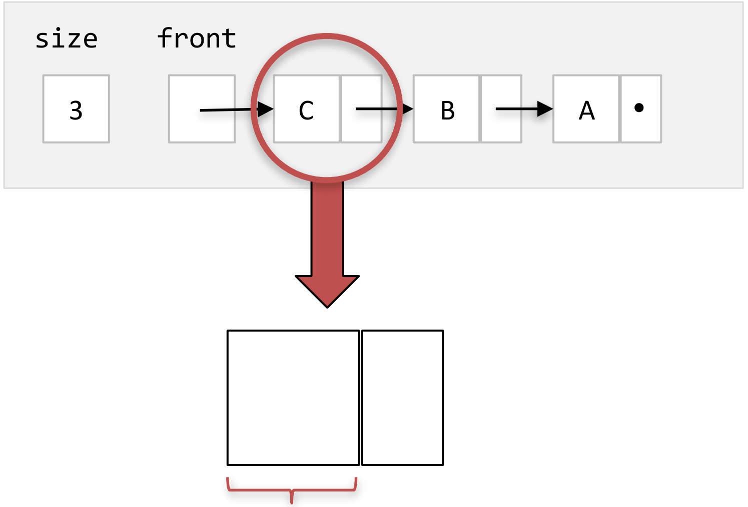
LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {




      . . .
   }

}
```



LinkedBag

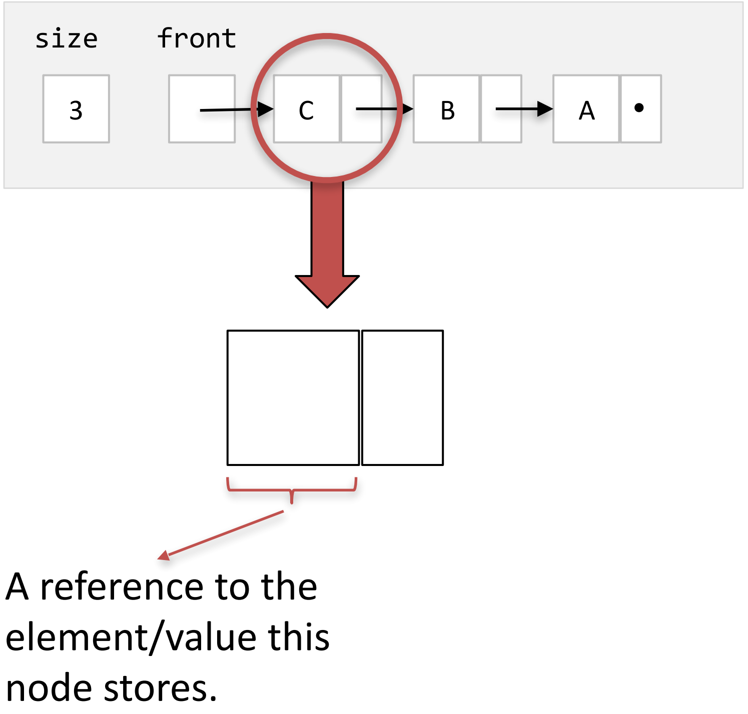
```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {



      . . .
   }

}
```



A reference to the element/value this node stores.

LinkedBag

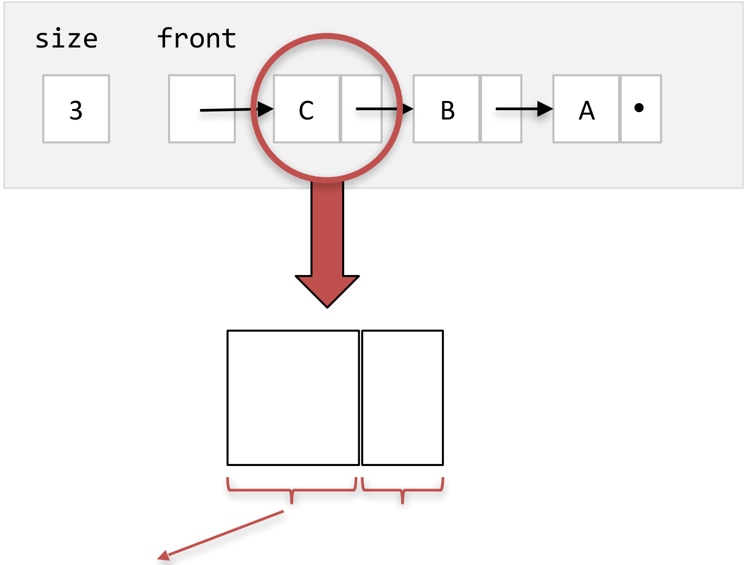
```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {

      private T element;


      . . .
   }

}
```



A reference to the element/value this node stores.

LinkedBag

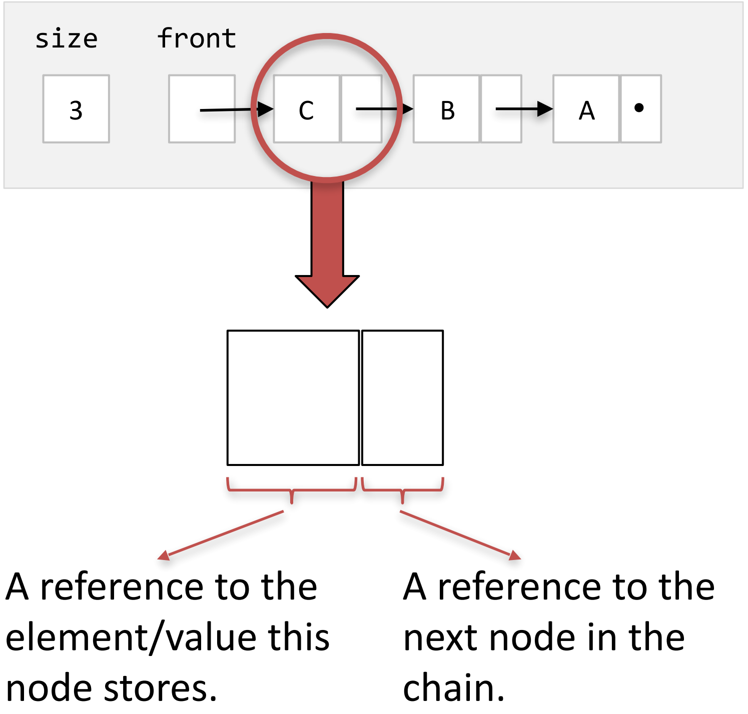
```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {

      private T element;


      . . .
   }

}
```



A reference to the element/value this node stores.

LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {

      private T element;


      . . .
   }

}
```

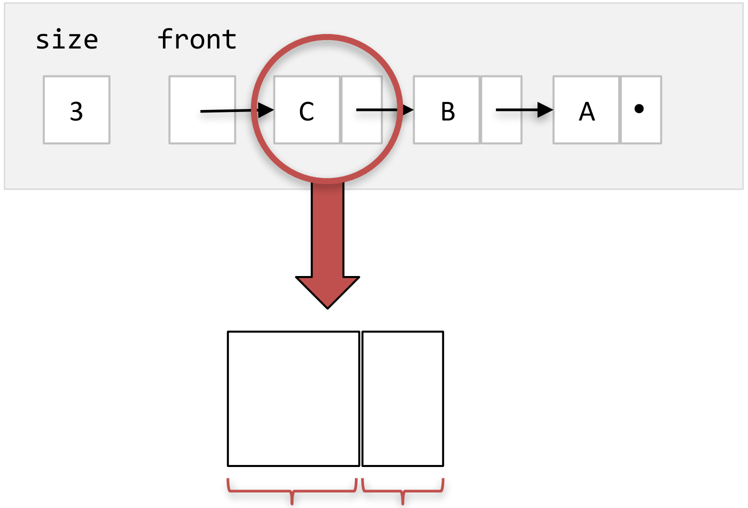


A reference to the element/value this node stores.

A reference to the next node in the chain.

LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {
      private T element;

      private Node next;
      . . .
   }

}
```



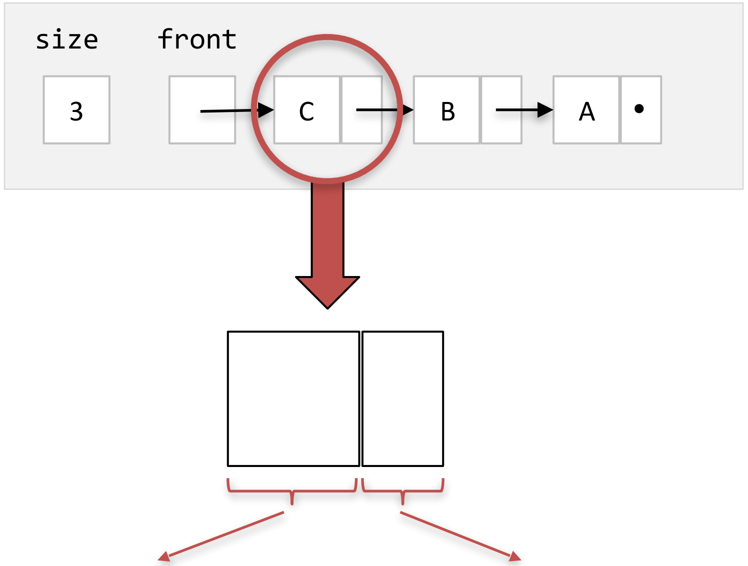
A reference to the element/value this node stores.

A reference to the next node in the chain.

LinkedBag

```
public class LinkedBag<T> implements Bag<T> {

   private Node front;
   private int size;
   . . .

   private class Node {
      private T element;
      private Node next;
      . . .
   }

}
```

A recursive structure [more to come…]



A reference to the element/value this node stores.

A reference to the next node in the chain.

The Node class

```
private class Node {
   private Object element;
   private Node next;

   public Node(Object e) {
      element = e;
   }

   public Node(Object e, Node n) {
      element = e;
      next = n;
   }
}
```

Constructors, garbage

```
n = new Node(1);

n = new Node(2, n);

n = new Node(3);

n = null;
```

The Node class

```
private class Node {
   private Object element;
   private Node next;

   public Node(Object e) {
      element = e;
   }

   public Node(Object e, Node n) {
      element = e;
      next = n;
   }
}
```

Basic linking

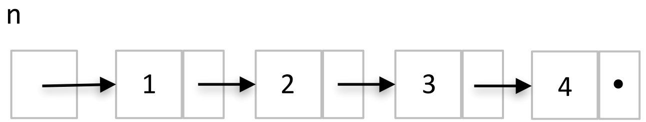
```
n = new Node(1);
n = new Node(2, n);
n.next = new Node(3, n.next);
```

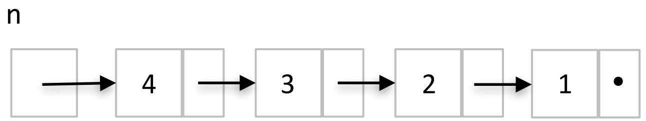
```
n = new Node(1, new Node(2));
n.next.next = new Node(3, null);
n = new Node(4, n.next);
```

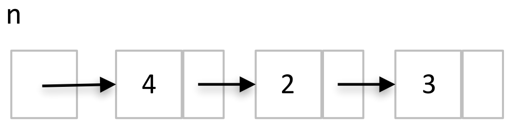
Participation

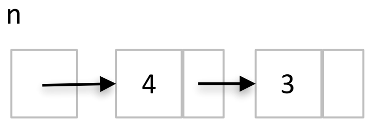
Q: Which chain of nodes is created by the following code?

```
n = new Node(1);

n.next = new Node(2, new Node(3));

n = new Node(4, n.next.next);
```

A.


B.


C.


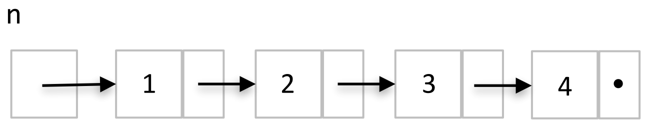
D.


Participation

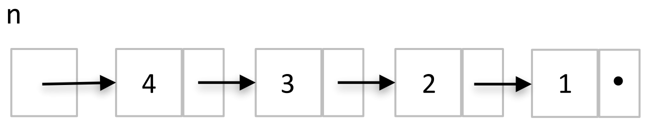
Q: Which chain of nodes is created by the following code?

```
n = new Node(1);

n.next = new Node(2, new Node(3));

n = new Node(4, n.next.next);
```

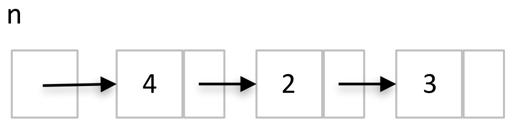
A.



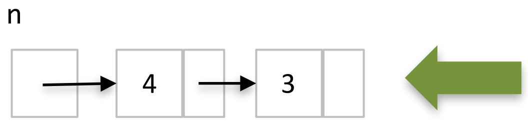
B.



C.

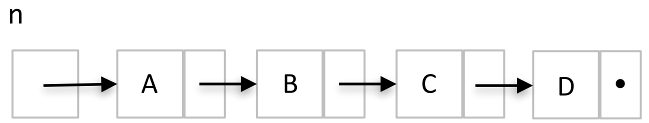


D.



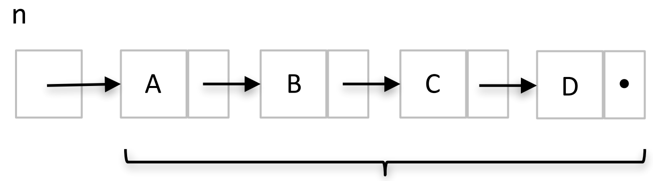
Calculating length

```
public int length(Node n) {





}
```



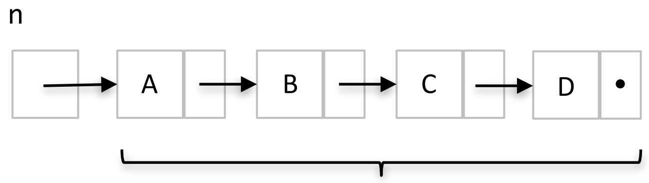
Calculating length

```
public int length(Node n) {





}
```



Calculating length

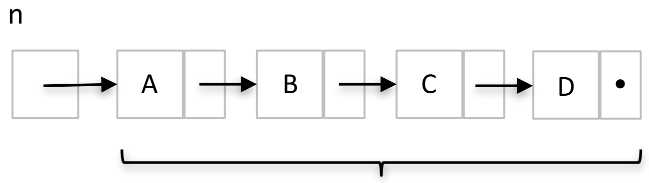
```
public int length(Node n) {




}
```



There are four nodes reachable from n, so the "length" of the chain is 4.

Calculating length

```
public int length(Node n) {





}
```

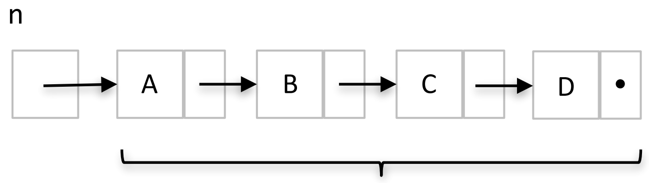


There are four nodes reachable from n, so the “length” of the chain is 4.

What solution pattern can we apply here?

Calculating length

```
public int length(Node n) {

        Linear scan

}
```

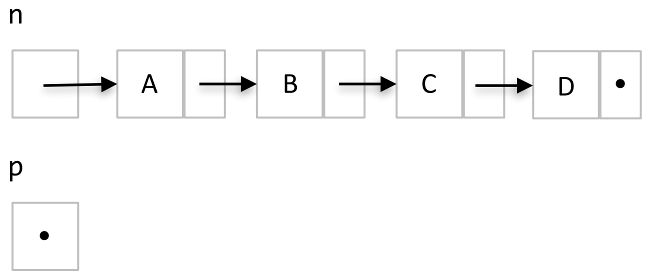


There are four nodes reachable from n, so the "length" of the chain is 4.

What solution pattern can we apply here?

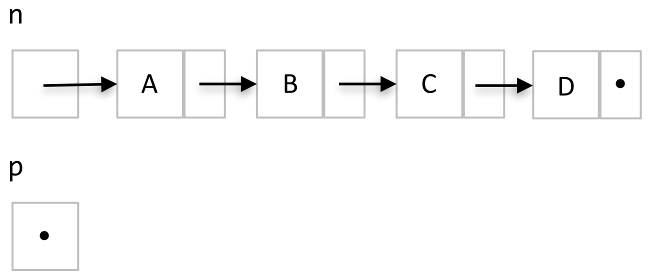
Calculating length

```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



Calculating length

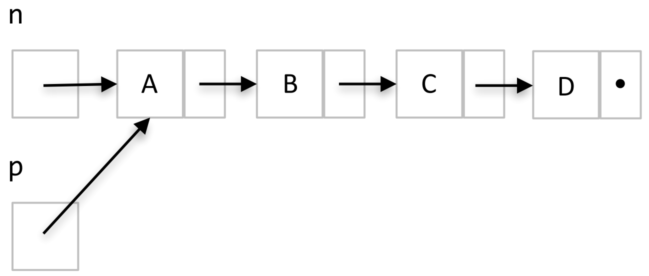
```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



This is a common **traversal** pattern that you will use in many different situations when you have perform a linear scan on a chain of nodes.

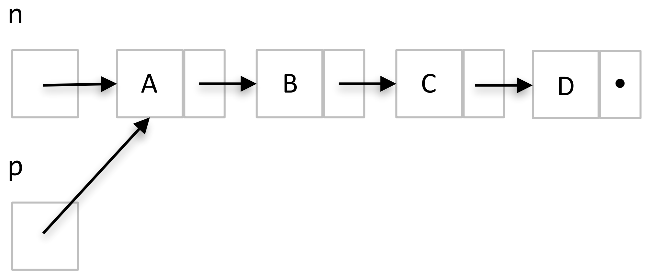
Calculating length

```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



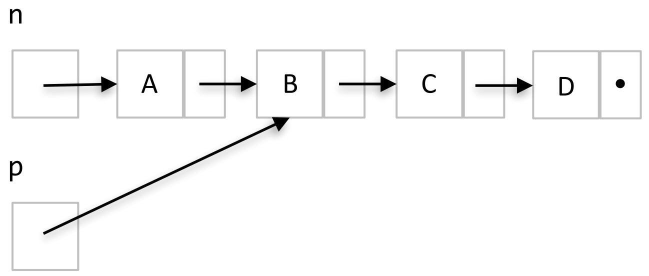
Calculating length

```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



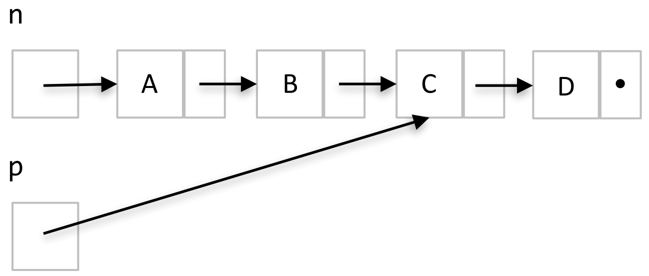
Calculating length

```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



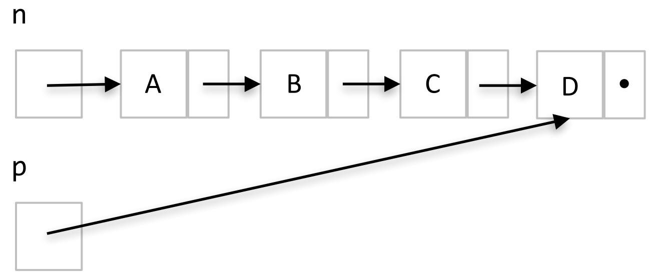
Calculating length

```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



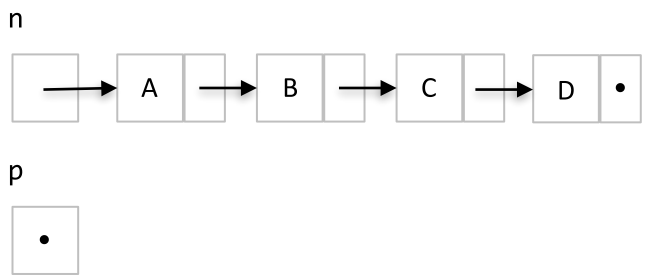
Calculating length

```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



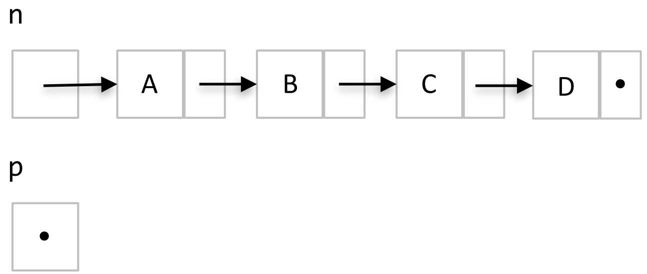
Calculating length

```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



Calculating length

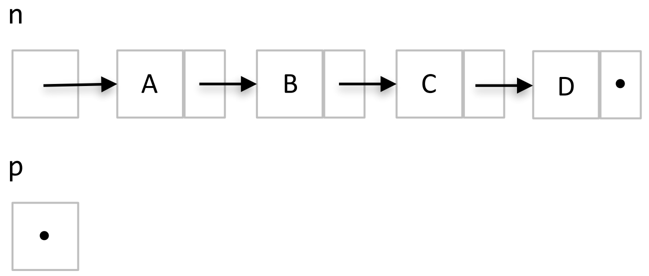
```
public int length(Node n) {
   Node p = n;

   while (p != null) {

      p = p.next;
   }

}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

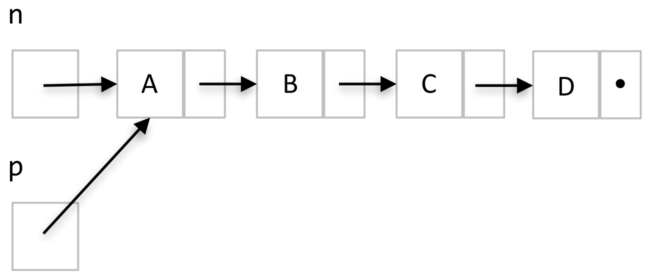
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

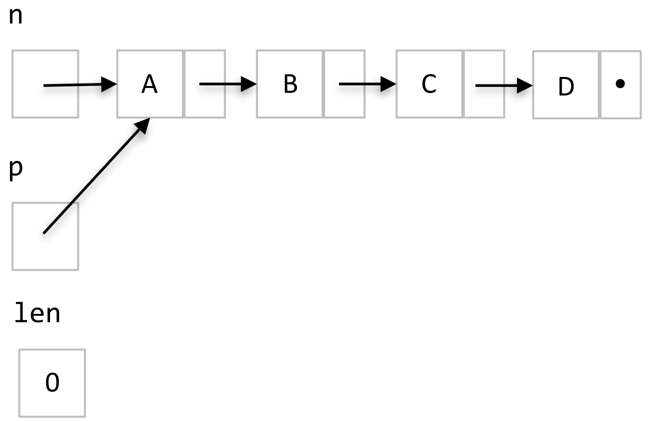
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

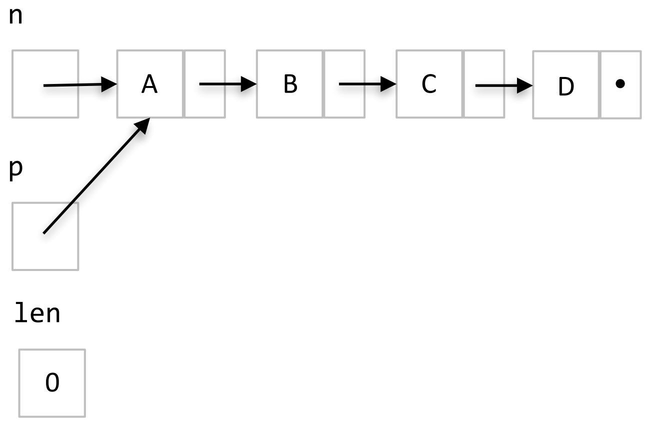
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```

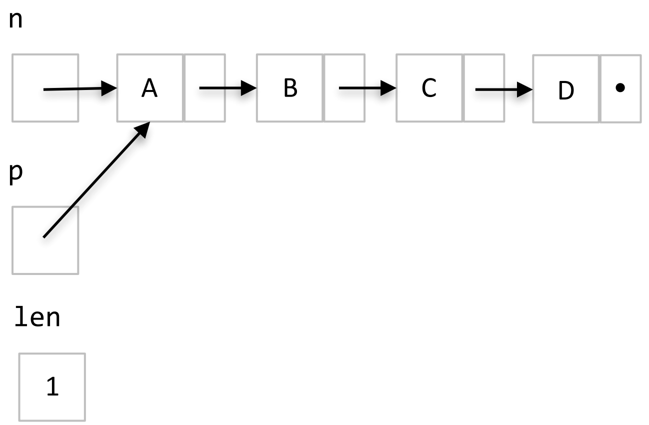


To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

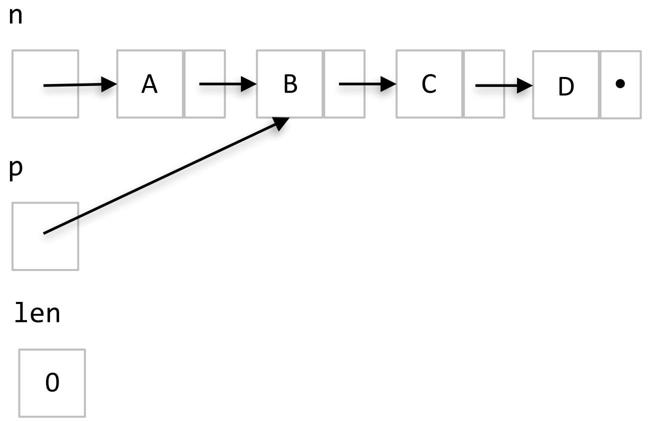
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```

To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.



Calculating length

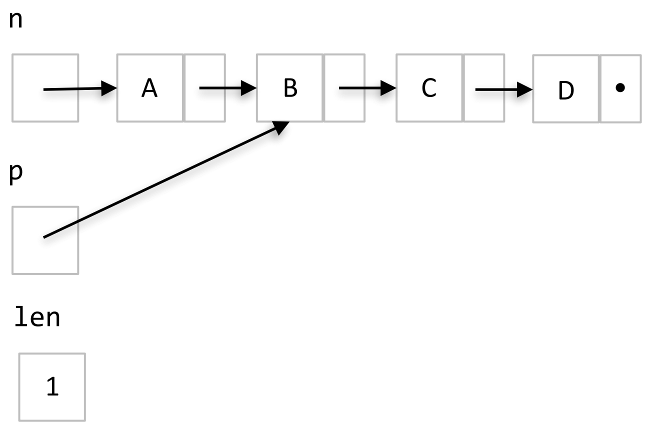
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

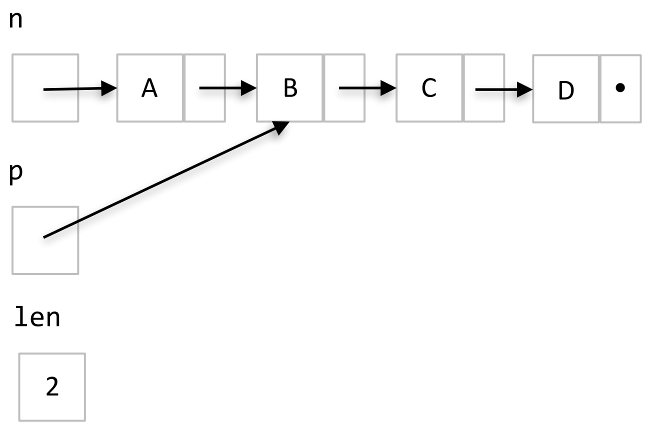
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

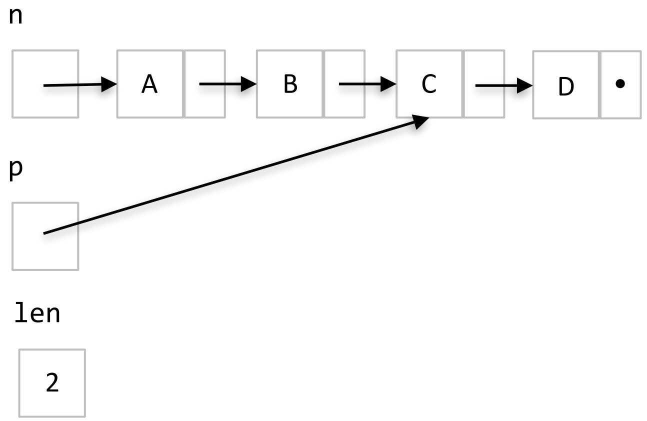
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

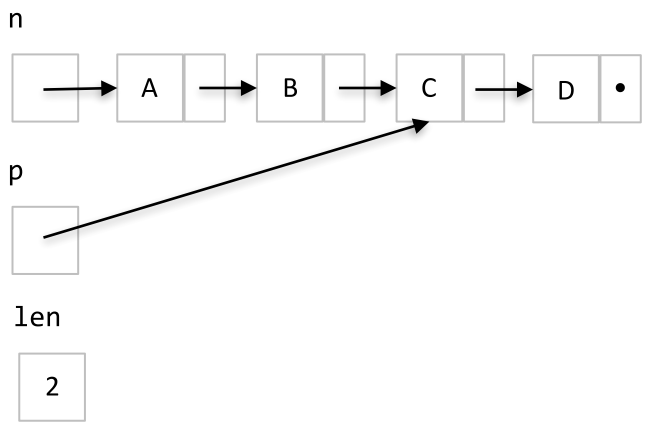
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

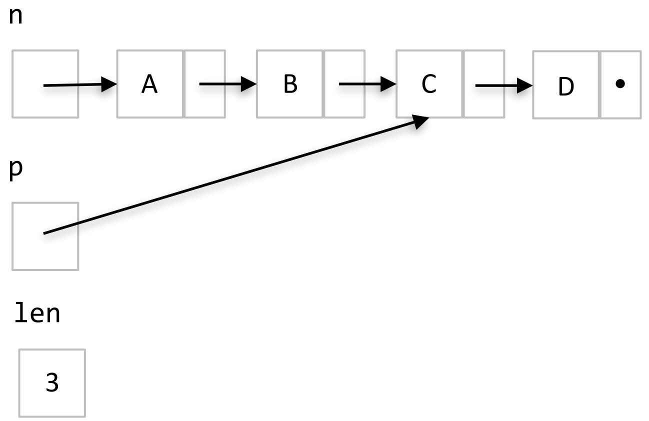
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

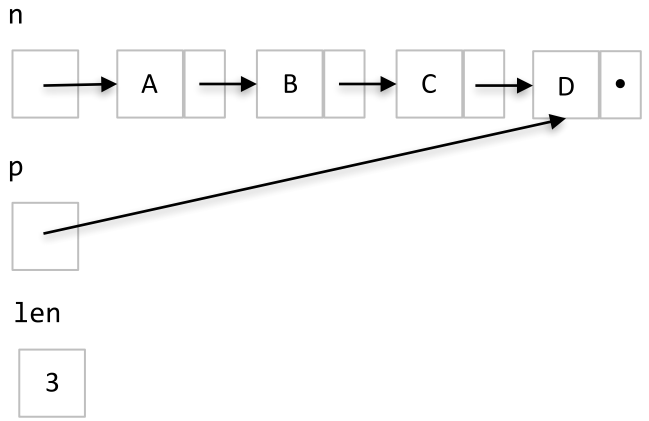
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

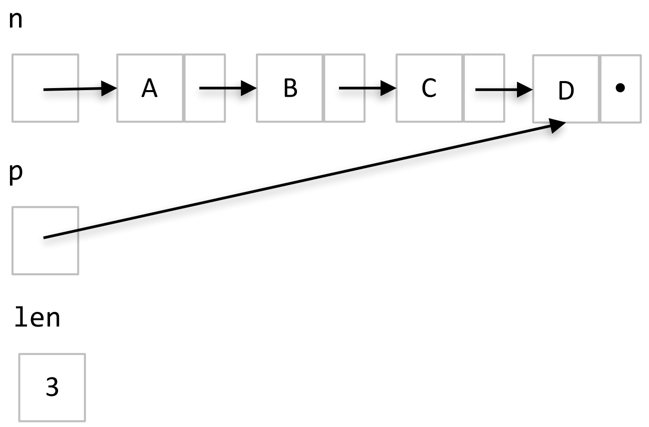
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

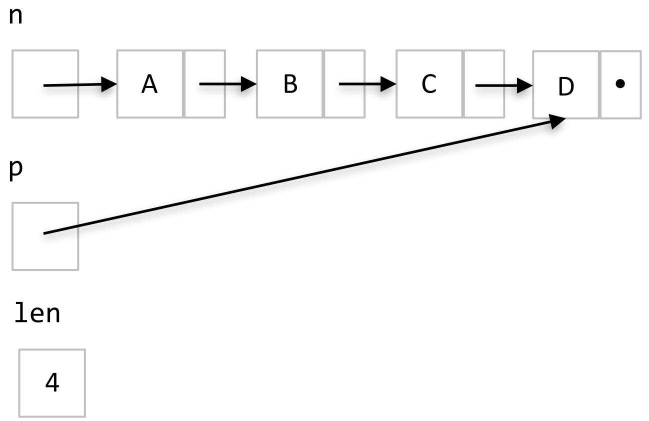
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

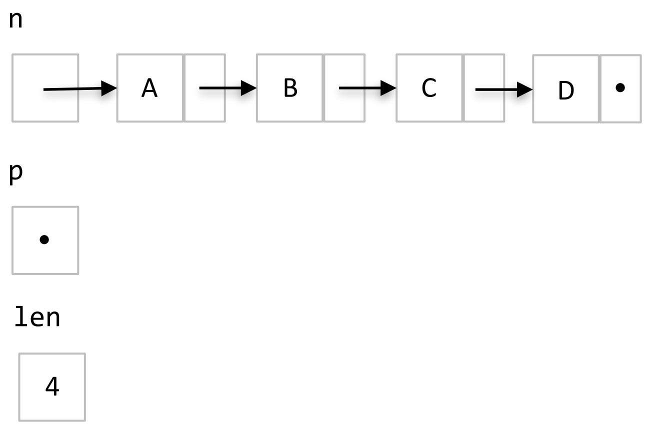
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

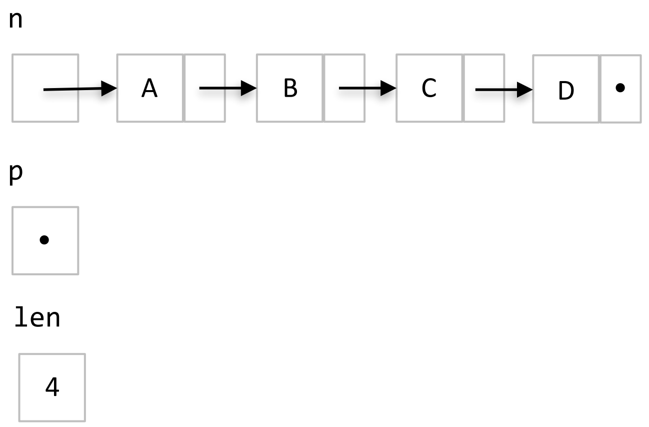
```java
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

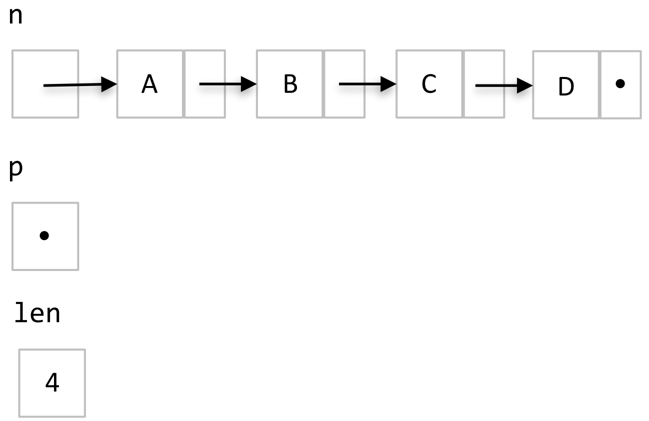
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

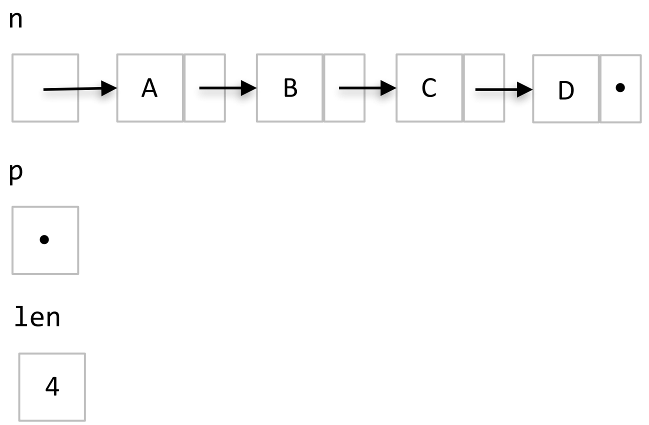
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

Calculating length

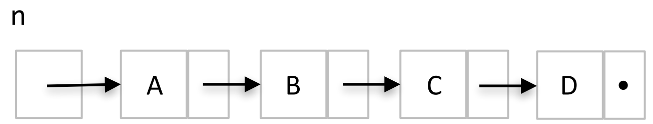
```
public int length(Node n) {
   Node p = n;
   int len = 0;
   while (p != null) {
      len++;
      p = p.next;
   }
   return len;
}
```



To compute the length of the pointer chain, simply add the statements to count each node that is accessed during the linear scan.

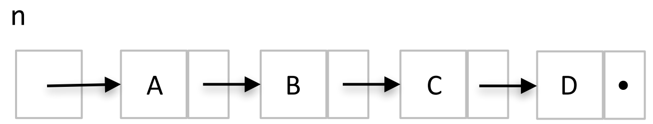
Linear search

```
public boolean contains(Node n, Object target) {







}
```



Linear search

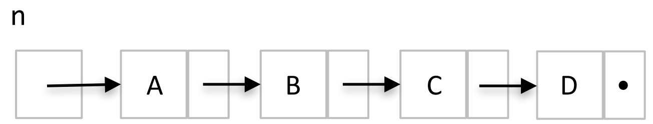
```
public boolean contains(Node n, Object target) {
    Node p = n;
    while (p != null) {



        p = p.next;
    }

}
```



Linear scan pattern

Linear search

```
public boolean contains(Node n, Object target) {
   Node p = n;
   while (p != null) {
      if (p.element.equals(target)) {
         return true;
      }
      p = p.next;
   }
   return false;
}
```



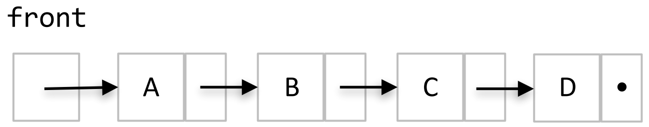
Linear scan pattern

+

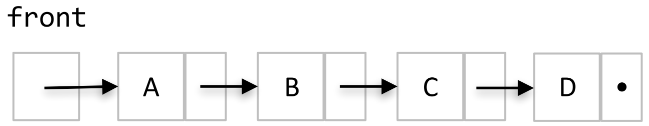
Problem-specific code

Inserting nodes

Inserting a new first node

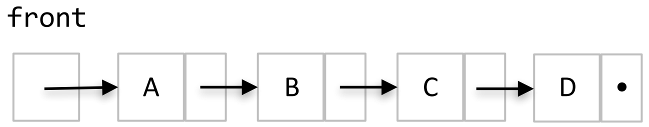


Inserting a new node somewhere else

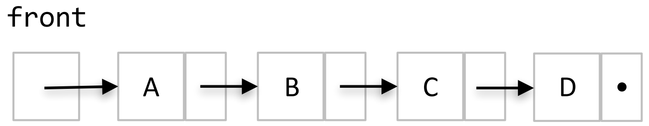


Inserting nodes

Inserting a new first node



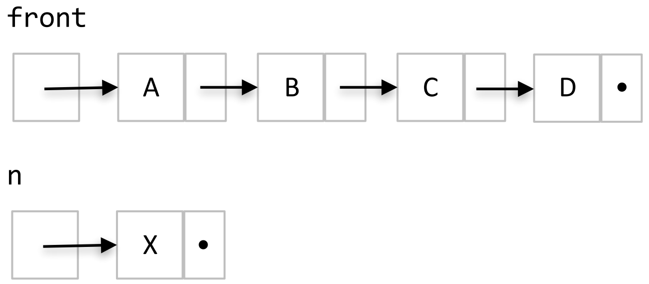
Inserting a new node somewhere else



```
Node n = new Node(“X”);
```

Inserting nodes

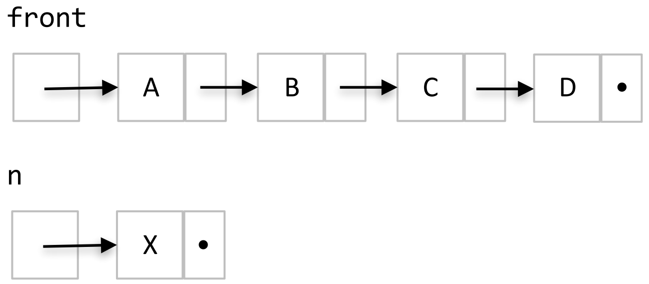
Inserting a new first node



```
Node n = new Node("X");
```

Inserting nodes

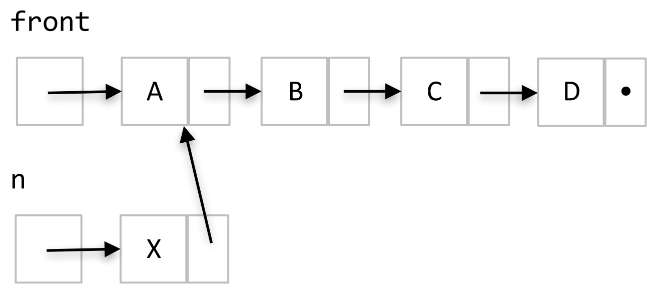
Inserting a new first node



```
Node n = new Node("X");


if (inserting a new first node) {


}
```

Inserting nodes

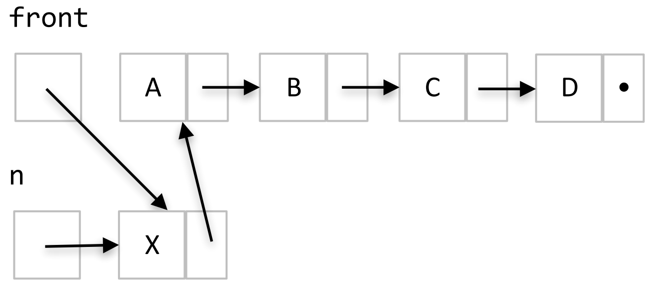
Inserting a new first node



```
Node n = new Node("X");


if (inserting a new first node) {
   n.next = front;

}
```

Inserting nodes

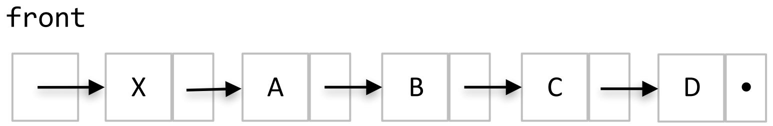
Inserting a new first node



```
Node n = new Node(“X”);


if (inserting a new first node) {
   n.next = front;
   front = n;
}
```

Inserting nodes

Inserting a new first node

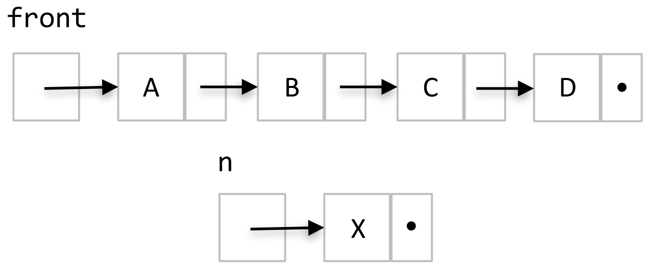


```
Node n = new Node(“X”);


if (inserting a new first node) {
   n.next = front;
   front = n;
}
```

Inserting nodes

```
Node n = new Node(“X”);


if (inserting a new first node) {
   n.next = front;
   front = n;
}


else {




}
```

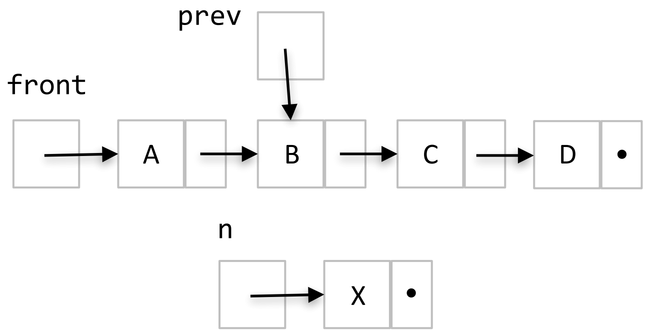
Inserting a new node somewhere else



Inserting nodes

```
Node n = new Node("X");


if (inserting a new first node) {
   n.next = front;
   front = n;
}


else {
   Node prev;
   // find the right spot with prev


}
```

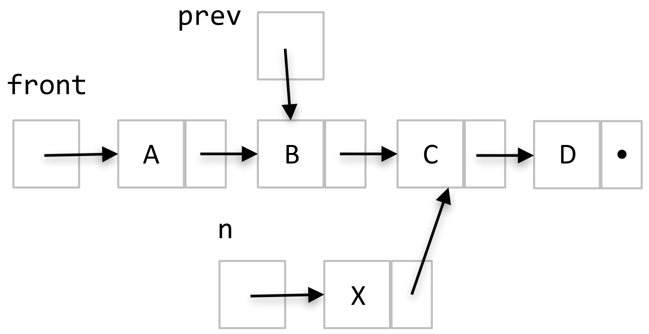
Inserting a new node somewhere else



Inserting nodes

```
Node n = new Node(“X”);


if (inserting a new first node) {
   n.next = front;
   front = n;
}


else {
   Node prev;
   // find the right spot with prev
   n.next = prev.next;

}
```

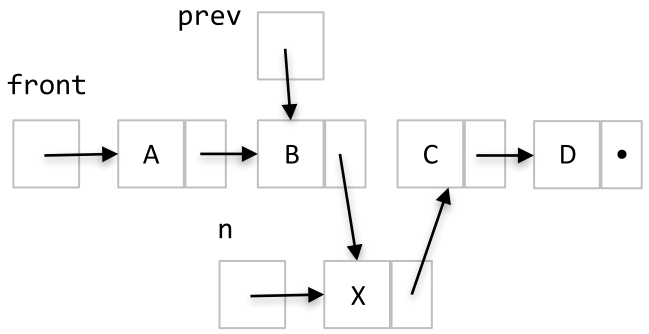
Inserting a new node somewhere else



Inserting nodes

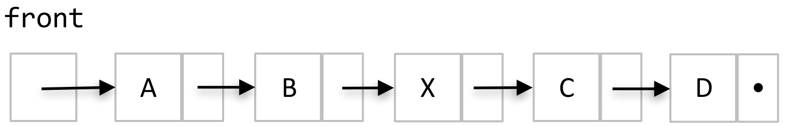
```
Node n = new Node(“X”);


if (inserting a new first node) {
   n.next = front;
   front = n;
}


else {
   Node prev;
   // find the right spot with prev
   n.next = prev.next;
   prev.next = n;
}
```

Inserting a new node somewhere else



Inserting nodes

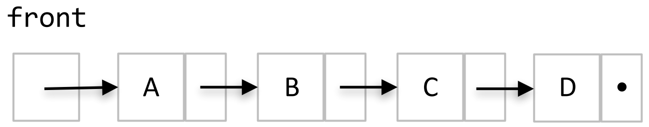
Inserting a new node somewhere else



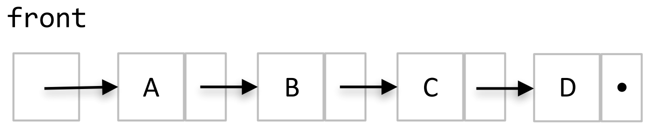
```
Node n = new Node("X");


if (inserting a new first node) {
   n.next = front;
   front = n;
}


else {
   Node prev;
   // find the right spot with prev
   n.next = prev.next;
   prev.next = n;
}
```

Deleting nodes

Deleting the first node



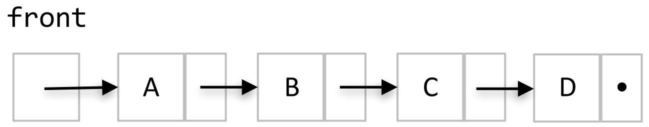
Deleting any other node



```
if (deleting the first node) {

}

else {


}
```

Deleting nodes

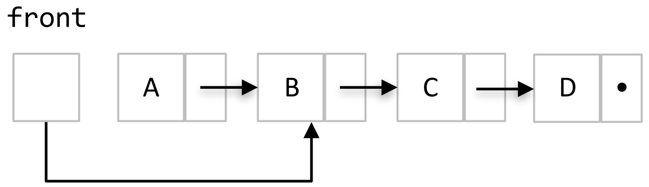
Deleting the first node



```
if (deleting the first node) {
   front = front.next;

}


else {




}
```

Deleting nodes

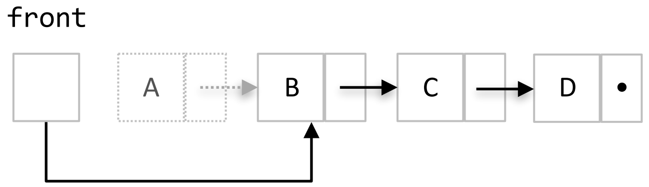
Deleting the first node



```
if (deleting the first node) {
   front = front.next;
}



else {




}
```

Deleting nodes

Deleting the first node

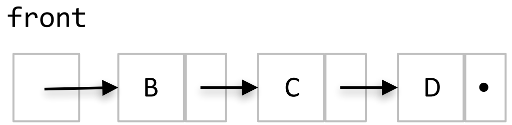


The node containing A is now garbage.

```
if (deleting the first node) {
   front = front.next;
}



else {




}
```

Deleting nodes

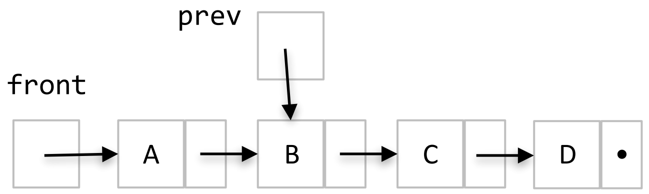
Deleting the first node



```
if (deleting the first node) {
   front = front.next;
}



else {




}
```

Deleting nodes

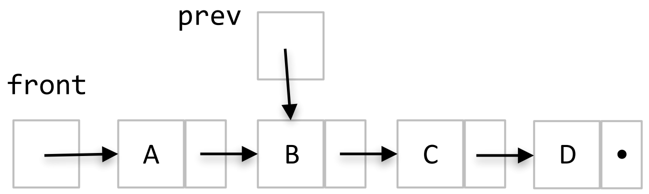
Deleting any other node



```
if (deleting the first node) {
   front = front.next;
}



else {
   Node prev;
   // find the right spot with prev


}
```

Deleting nodes

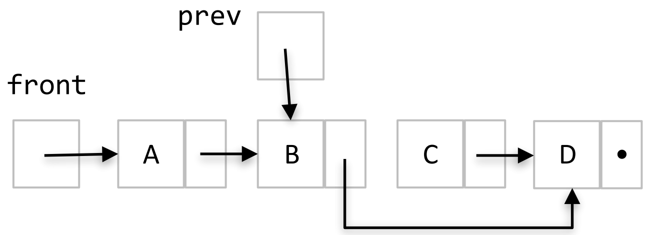
Deleting any other node



```
if (deleting the first node) {
   front = front.next;
}



else {
   Node prev;
   // find the right spot with prev
   prev.next = prev.next.next;

}
```

Deleting nodes

Deleting any other node

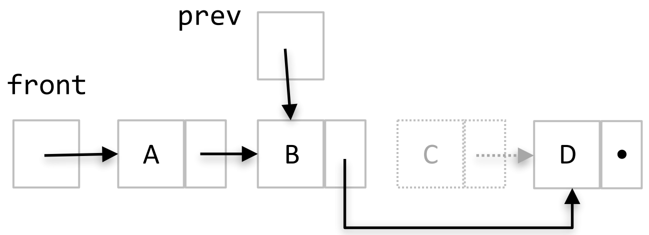


```
if (deleting the first node) {
   front = front.next;
}



else {
   Node prev;
   // find the right spot with prev
   prev.next = prev.next.next;

}
```

Deleting nodes

The node containing C is now garbage.

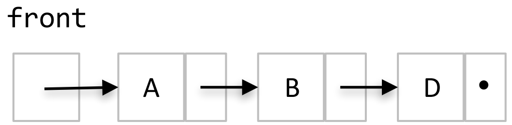
Deleting any other node



```
if (deleting the first node) {
   front = front.next;
}



else {
   Node prev;
   // find the right spot with prev
   prev.next = prev.next.next;

}
```

Deleting nodes

Deleting any other node



```
if (deleting the first node) {
   front = front.next;
}



else {
   Node prev;
   // find the right spot with prev
   prev.next = prev.next.next;

}
```