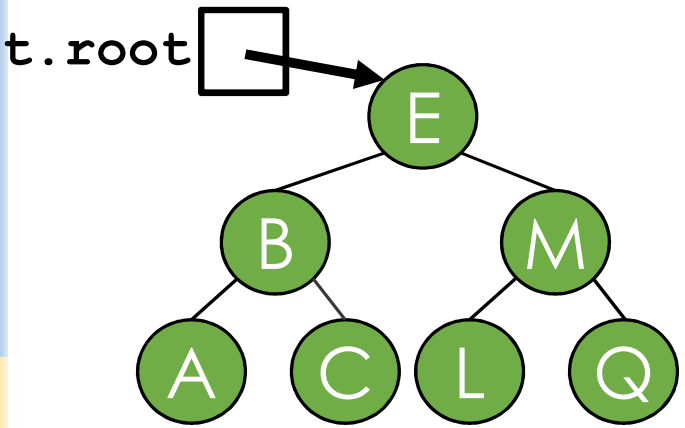
# Binary Search Trees: Search

Let's write it!

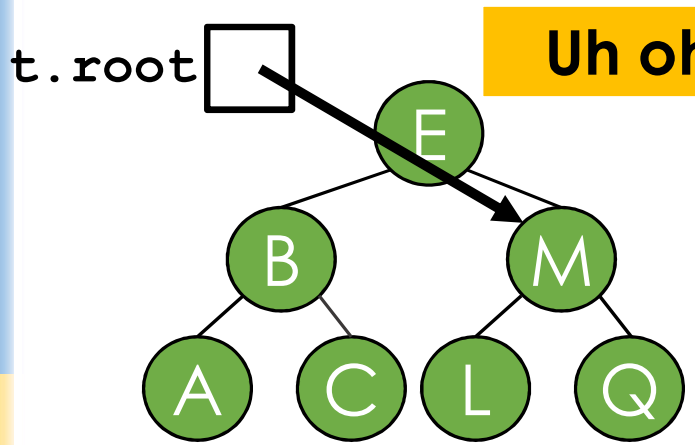
# Looking ahead…

- You will implement search in a different kind of tree in the project, but the ideas are the same




```
t.search('L')
```

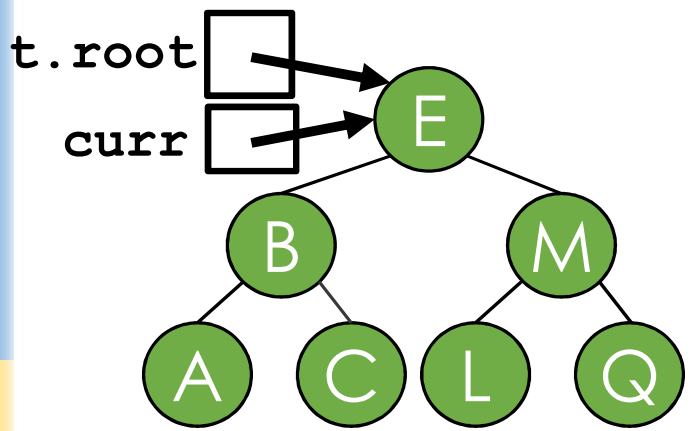
```
public boolean contains(E toFind)
{
```




**Uh oh!**

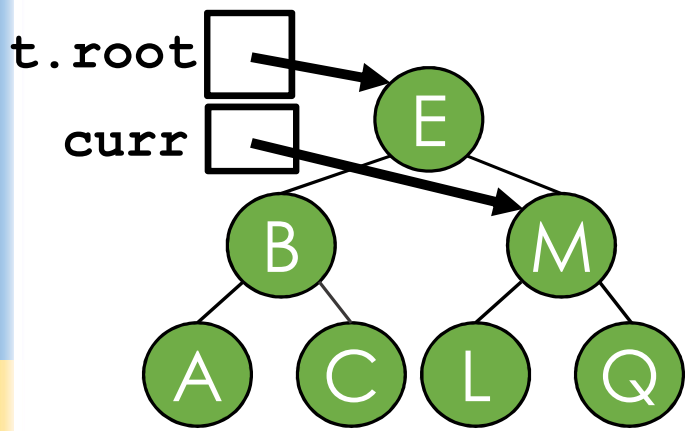
```
public boolean contains(E toFind)
{
```

```
t.search('L')
```



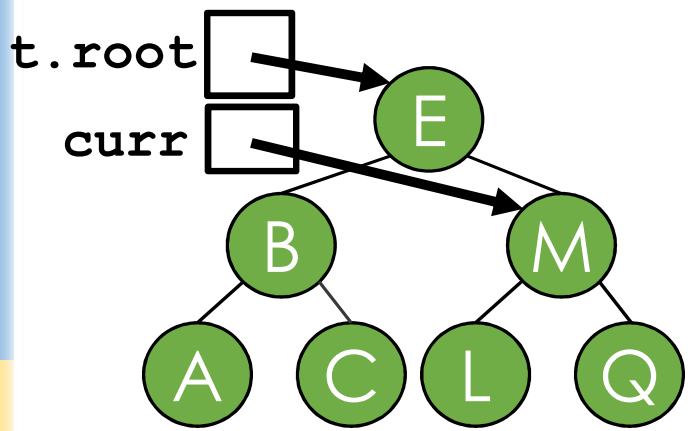

```
t.search('L')
```

```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;
```



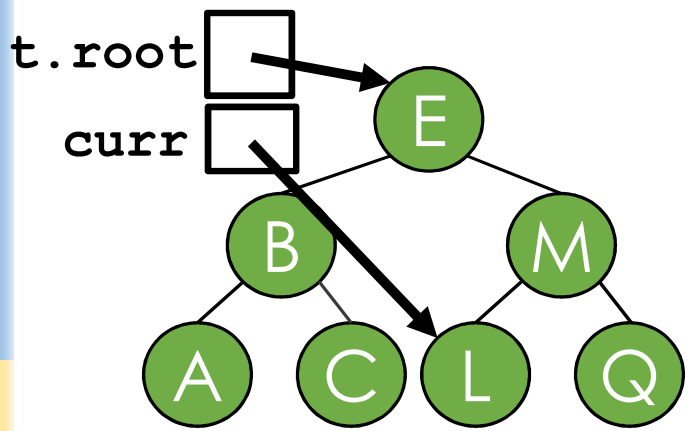

```
t.search('L')
```

```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;
```



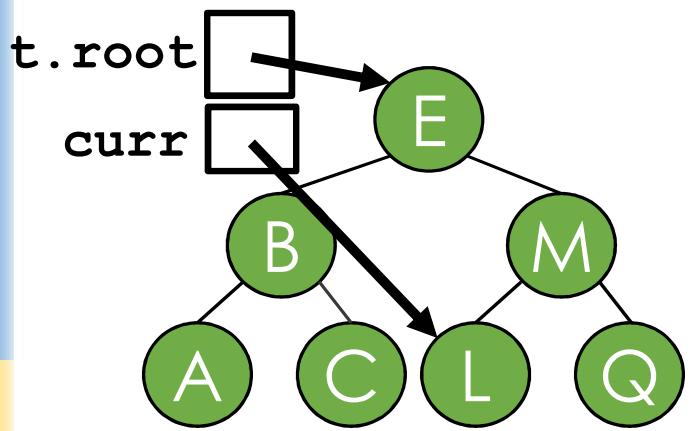

t.search('L')

```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;


  if (toFind < curr.getData())
    curr = curr.getLeft();
  else if (toFind > curr.getData())
    curr = curr.getRight();
  else
```




```
t.search('L')
```

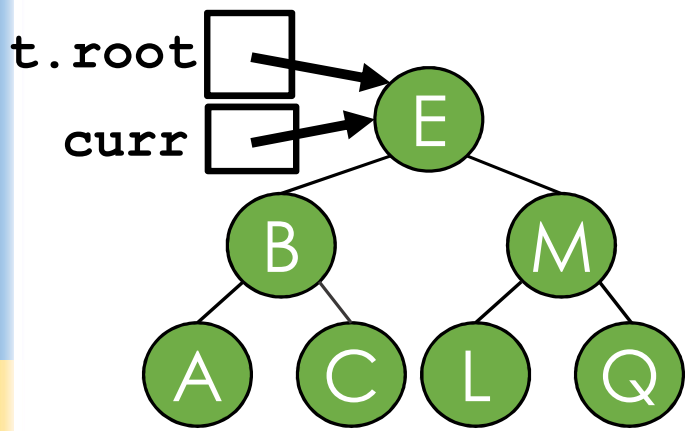
```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;


  if (toFind < curr.getData())
    curr = curr.getLeft();
  else if (toFind > curr.getData())
    curr = curr.getRight();
  else
```




```
t.search('L')
```

```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;


  if (toFind < curr.getData())
    cu
  else                          Data())
    cu
  else
    return true;
```

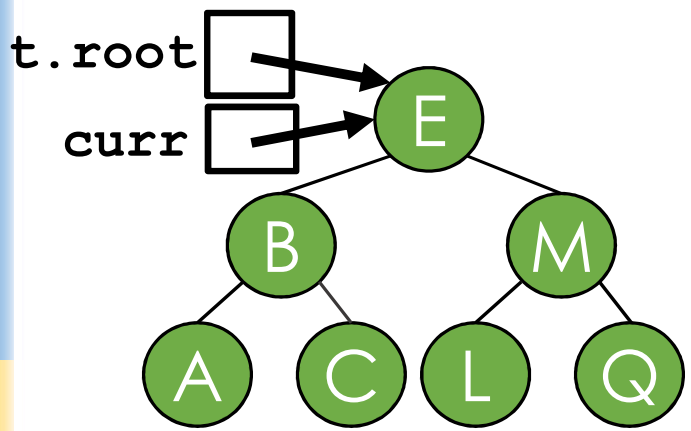
We need to do this over and over!




`t.search('L')`

```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;

  while (curr != null) {
    if (toFind < curr.getData())
      curr = curr.getLeft();
    else if (toFind > curr.getData())
      curr = curr.getRight();
    else
      return true;
  }
}
```

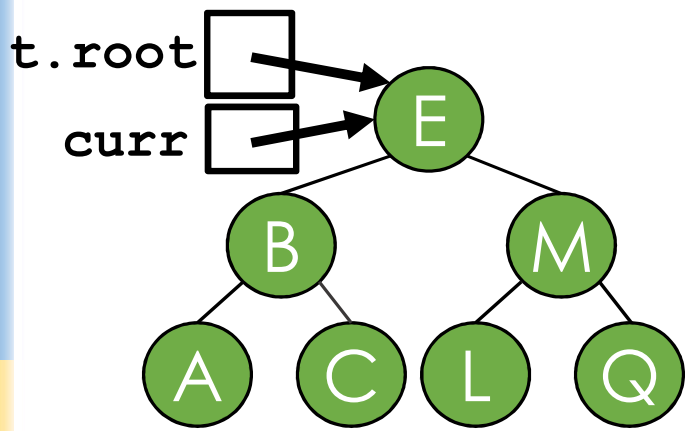
**Are we done?**




```
t.search('L')
```

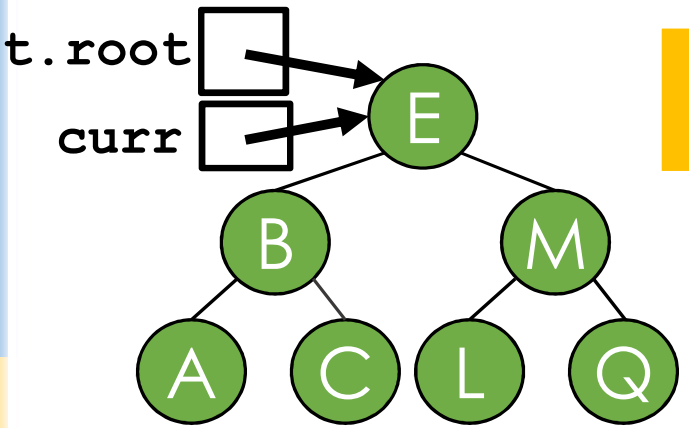
```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;

  while (curr != null) {
    if (toFind < curr.getData())
      curr = curr.getLeft();
    else if (toFind > curr.getData())
      curr = curr.getRight();
    else
      ret
  }
  return false;
}
```

**Doesn't work with objects**




```
t.search('L')
```

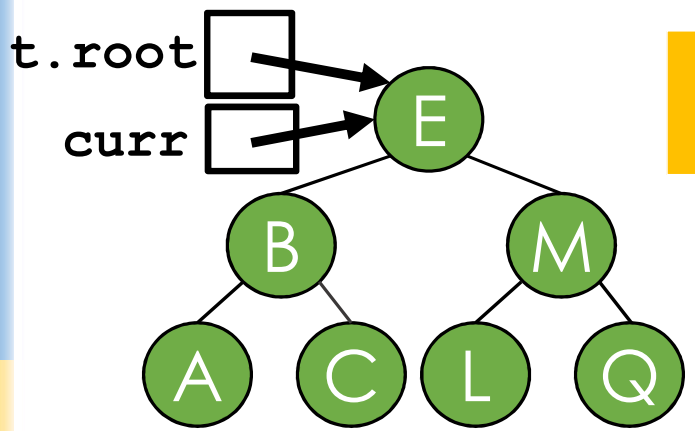
```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;
  int comp =
    toFind.compareTo(curr.getData());
  while (curr != null) {
    if (comp < 0)
      curr = curr.getLeft();
    else if (comp > 0)
      curr = curr.getRight();
    else
      return true;
  }
  return false;
}
```




```
t.search('L')
```

**if calling object is less than parameter, compareTo returns a value < 0**

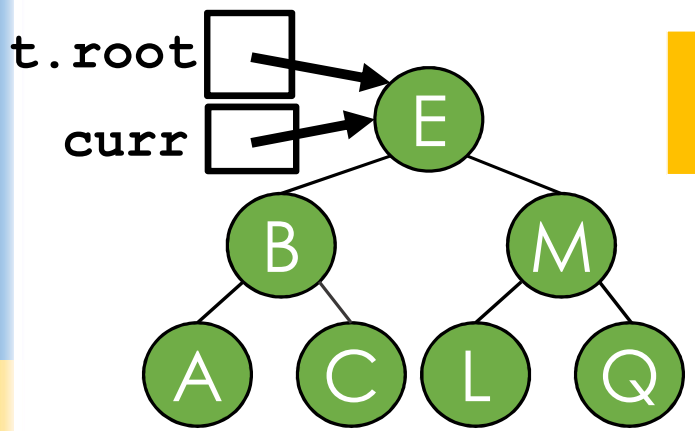
```
                                                    nd)
    TreeNode<E> curr = root;
    int comp =
      toFind.compareTo(curr.getData());
    while (curr != null) {
      if (comp < 0)
        curr = curr.getLeft();
      else if (comp > 0)
        curr = curr.getRight();
      else
        return true;
    }
    return false;
  }
```




```
t.search('L')
```

**if calling object is greater than parameter, compareTo returns a value > 0**

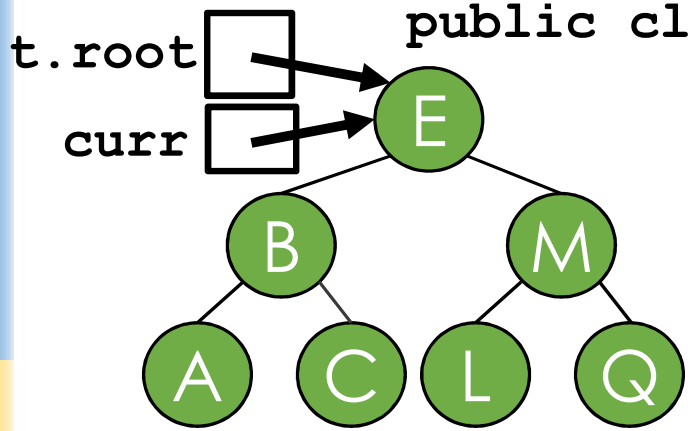
```
TreeNode<E> curr = root;
int comp =
  toFind.compareTo(curr.getData());
while (curr != null) {
  if (comp < 0)
    curr = curr.getLeft();
  else if (comp > 0)
    curr = curr.getRight();
  else
    return true;
}
return false;
}
```




```
t.search('L')
```

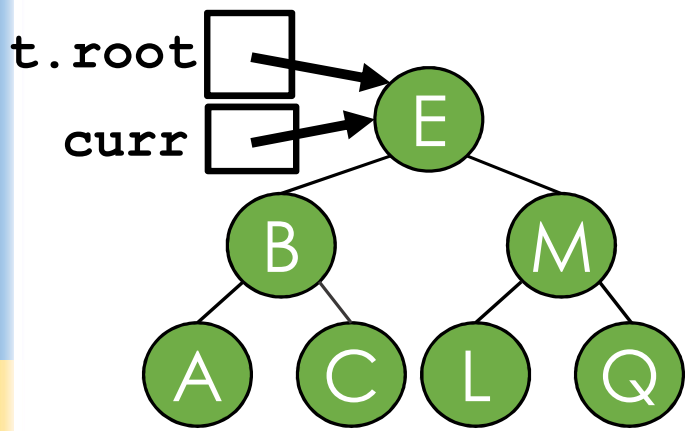
**if calling object is equal to parameter, compareTo returns 0**

```
)
    TreeNode<E> curr = root;
    int comp =
      toFind.compareTo(curr.getData());
    while (curr != null) {
      if (comp < 0)
        curr = curr.getLeft();
      else if (comp > 0)
        curr = curr.getRight();
      else
        return true;
    }
    return false;
  }
```




```
public class BST<E extends Comparable<? super E>>

public boolean contains(E toFind)
{
   …
```

```
t.search('L')
```




```
t.search('L')
```

```
public boolean contains(E toFind)
{
  TreeNode<E> curr = root;
  int comp =
    toFind.compareTo(curr.getData());
  while (curr != null) {
    if (comp < 0)
      curr = curr.getLeft();
    else if (comp > 0)
      curr = curr.getRight();
    else
      return true;
  }
  return false;
}
```

# Next step

- Insertion and deletion in a BST