Creating a Copy of a Linked List

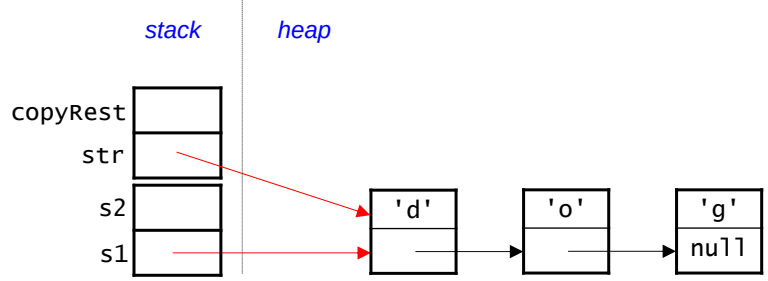
- `copy(str)` – create a copy of *the entire list* to which `str` refers
- Recursive approach:
  - base case: if `str` is empty, return `null`
  - else: – make a recursive call to copy the rest of the linked list
    – create and return a copy of the first node, with its next field pointing to the copy of the rest

```
public static StringNode copy(StringNode str) {
    if (str == null) {        // base case
        return null;
    }

    // make a recursive call to copy the rest of the list
    StringNode copyRest = copy(str.next);

    // create and return a copy of the first node,
    // with its next field pointing to the copy of the rest
    return new StringNode(str.ch, copyRest);
}
```

Tracing copy(): the initial call

- From a client: `StringNode s2 = StringNode.copy(s1);`

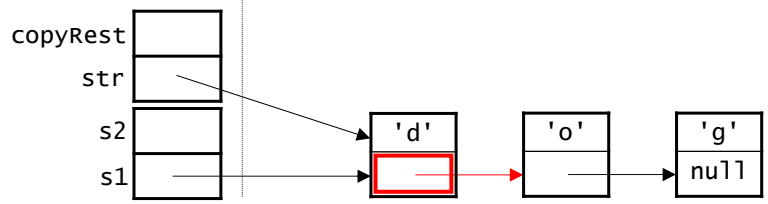
```
public static StringNode copy(StringNode str) {
    if (str == null) {
        return null;
    }

    StringNode copyRest = copy(str.next);
    return new StringNode(str.ch, copyRest);
}
```



Tracing copy(): the initial call

- From a client: `StringNode s2 = StringNode.copy(s1);`

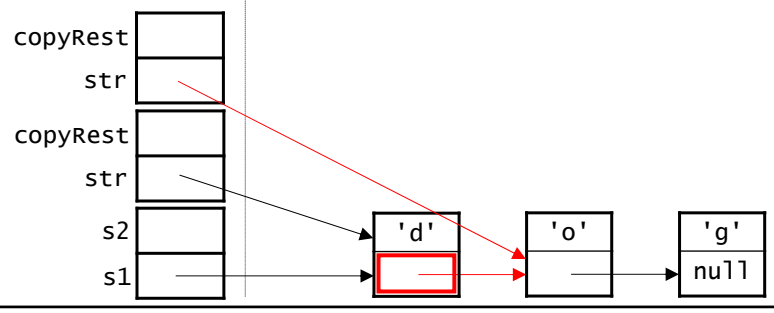
```
public static StringNode copy(StringNode str) {
    if (str == null) {
        return null;
    }
    StringNode copyRest = copy(str.next);
    return new StringNode(str.ch, copyRest);
}
```



Tracing copy(): the initial call

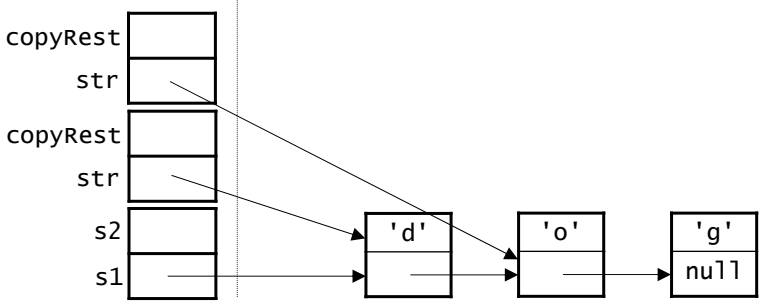
- From a client: `StringNode s2 = StringNode.copy(s1);`

```
public static StringNode copy(StringNode str) {
    if (str == null) {
        return null;
    }
    StringNode copyRest = copy(str.next);
    return new StringNode(str.ch, copyRest);
}
```



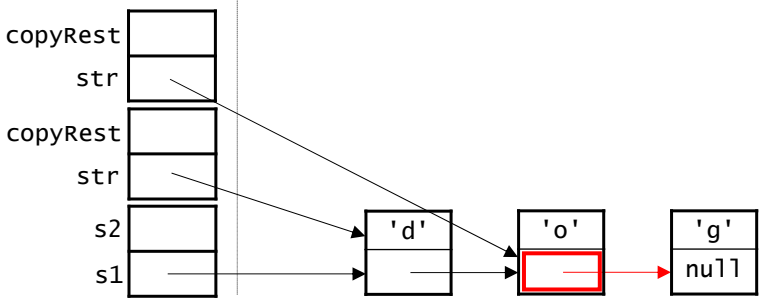
Tracing copy(): the recursive calls

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



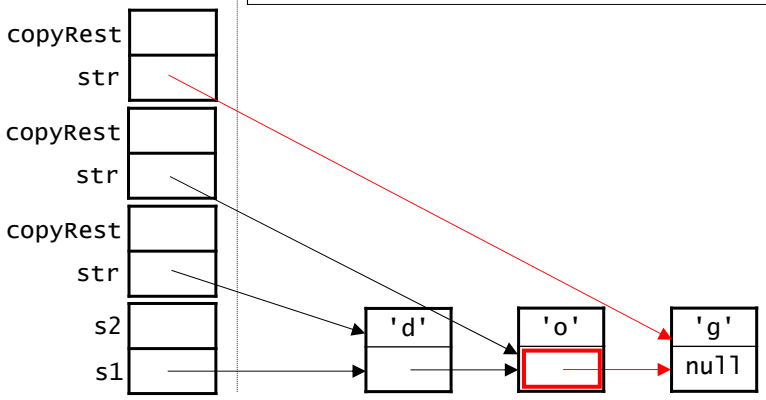
Tracing copy(): the recursive calls

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



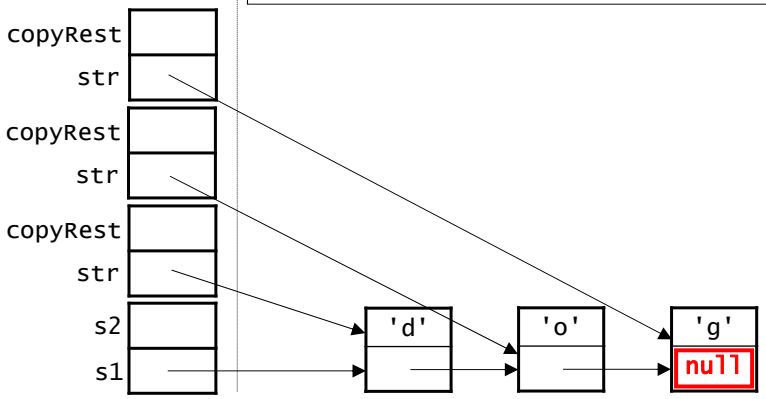
Tracing copy(): the recursive calls

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }
  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



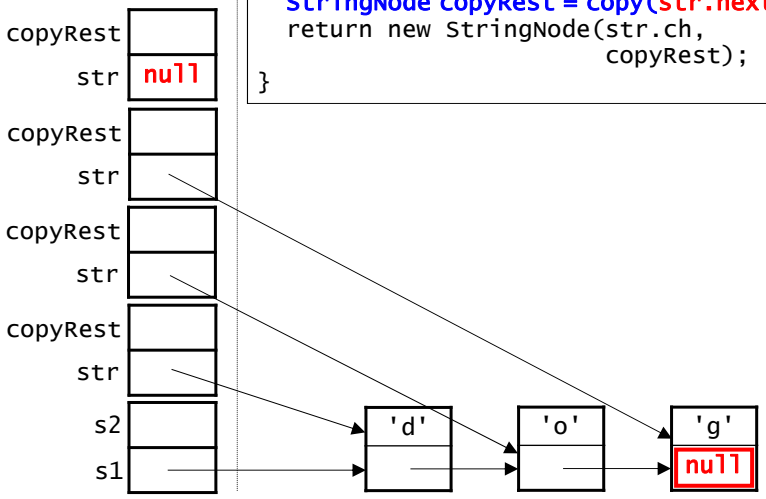
Tracing copy(): the recursive calls

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }
  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



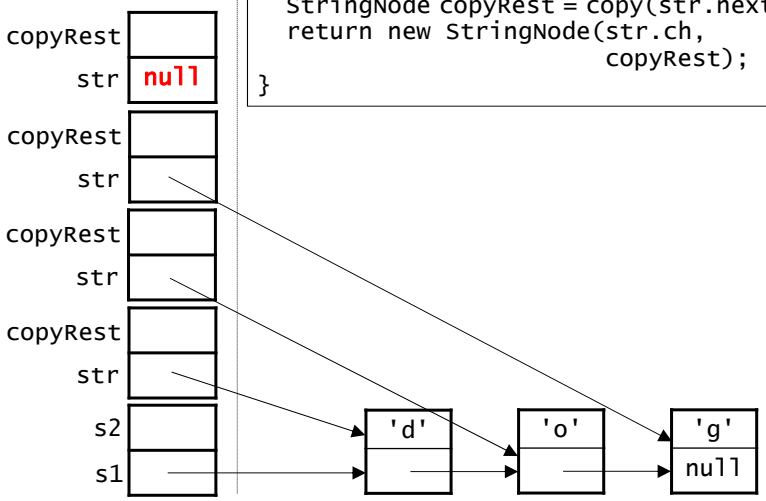
Tracing copy(): the recursive calls

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



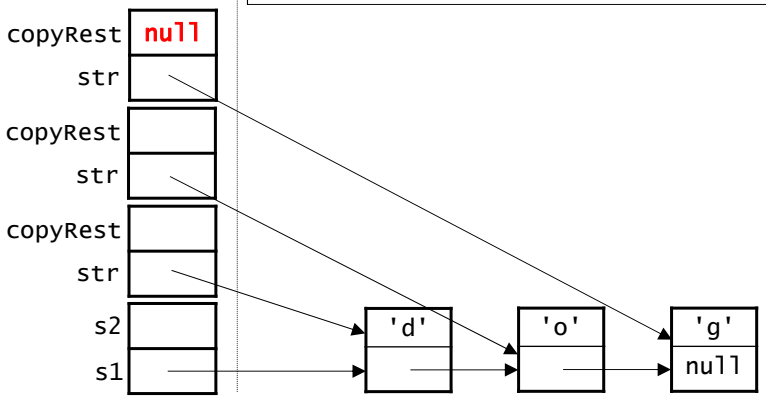
Tracing copy(): the base case

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



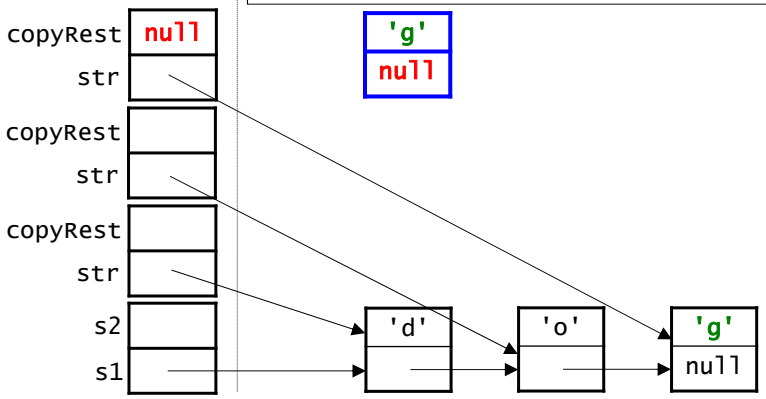
Tracing copy(): returning from the base case

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



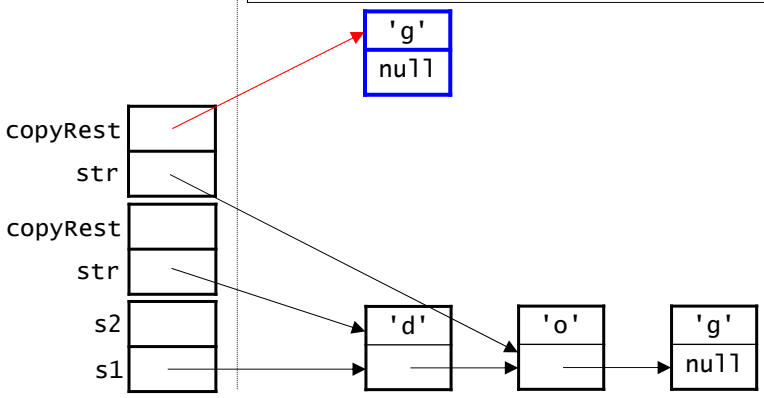
Tracing copy(): returning from the base case

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



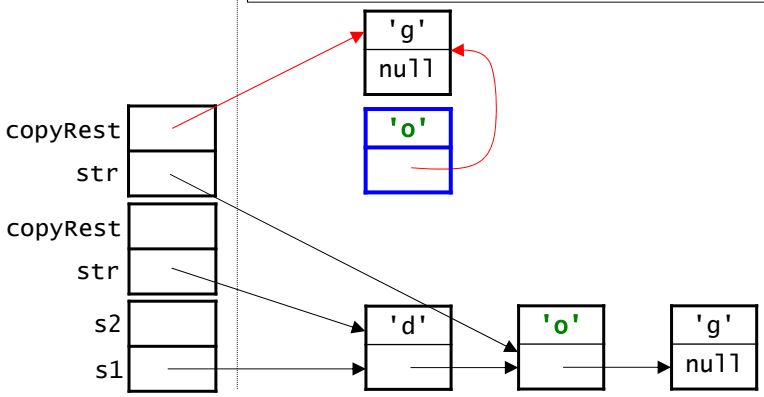
Tracing copy(): returning from the base case

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



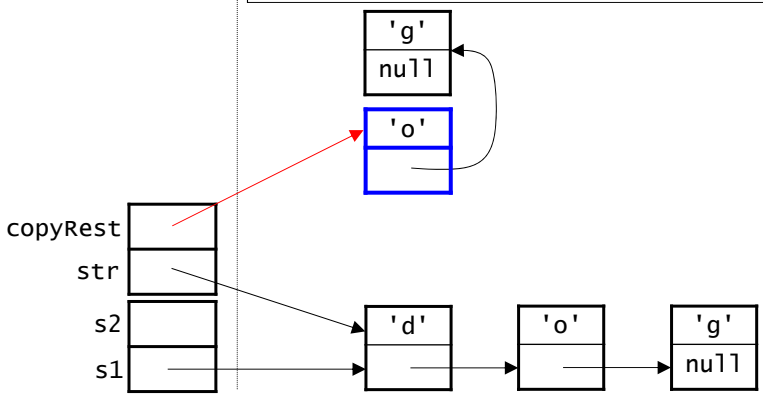
Tracing copy(): returning from the base case

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



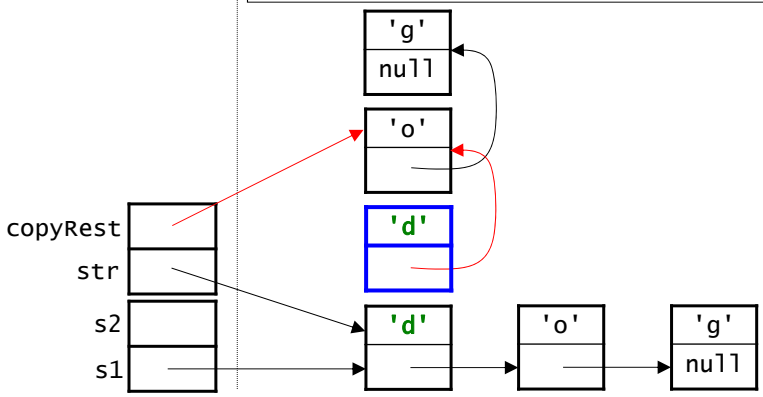
Tracing copy(): returning from the base case

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



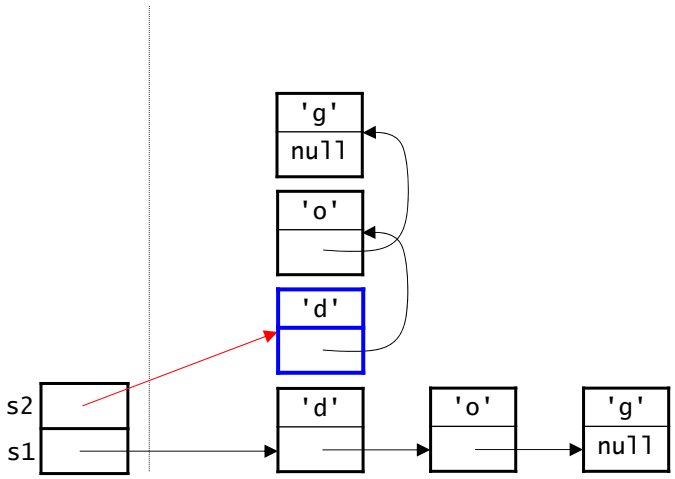
Tracing copy(): returning from the base case

```
public static StringNode copy(...) {
  if (str == null) {
      return null;
  }

  StringNode copyRest = copy(str.next);
  return new StringNode(str.ch,
                        copyRest);
}
```



Tracing copy(): returning from the base case

- From a client: `StringNode s2 = StringNode.copy(s1);`



Tracing copy(): Final Result

- `s2` now holds a reference to a linked list that is a copy of the linked list to which `s1` holds a reference.