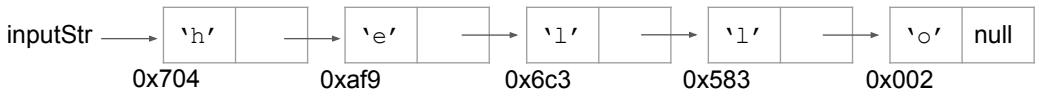
Section 5

CSCI E-22

Will Begin Shortly

Tracing through `StringNode.print()`

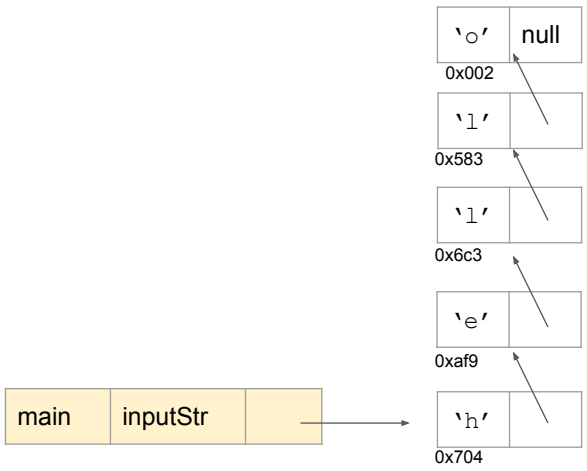
Let's take a look at the execution of the `StringNode.print()` method where the input list looks like this (the hexadecimal number next to each node is the memory location of that node):



We want to draw a diagram which shows how the stack evolves with each subsequent call, like we did on the written portion of Problem Set 1. The important things to track are the output so far, and the address the current `str` variable holds in each frame.

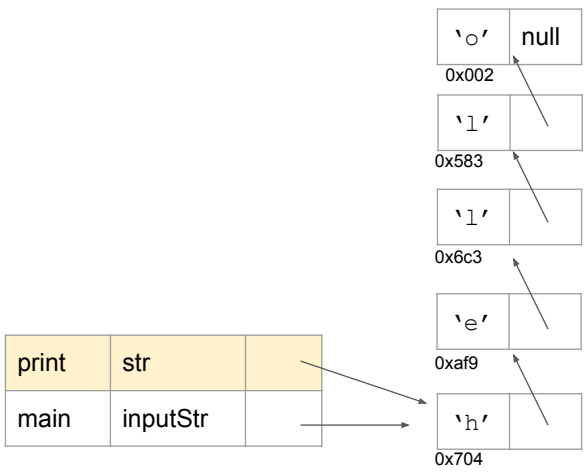
Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
 }
```



Tracing through `StringNode.print()`

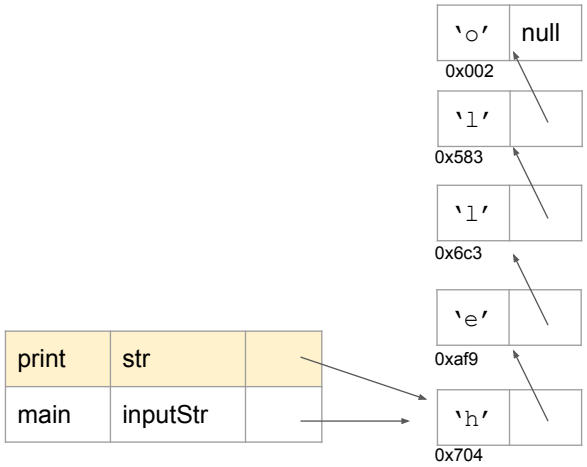
```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
 }
```



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

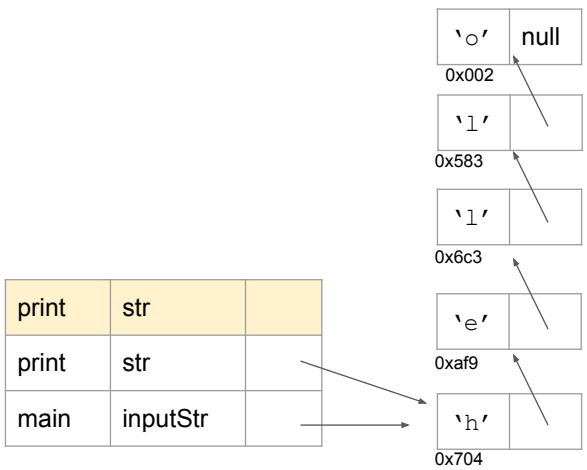
Output: `h`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

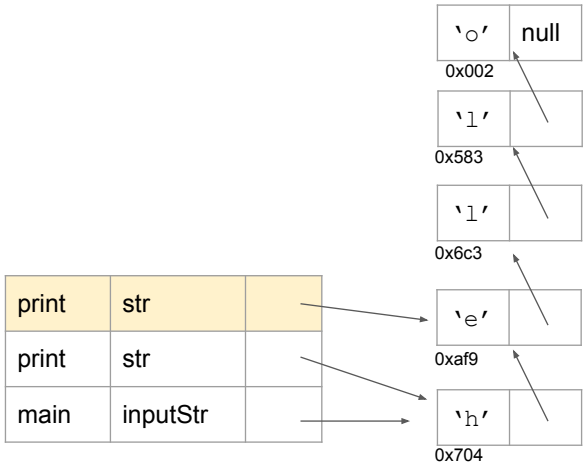
Output: `h`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

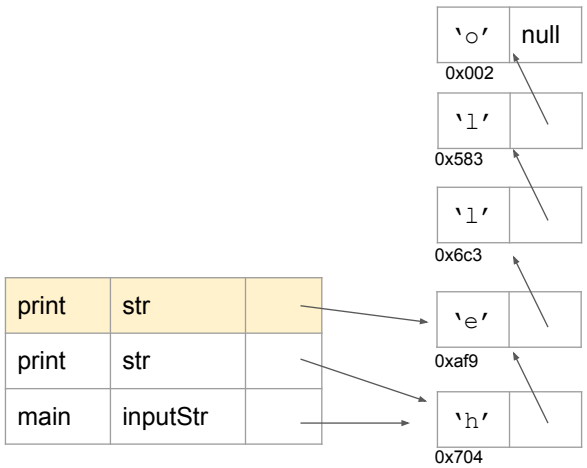
Output: `h`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

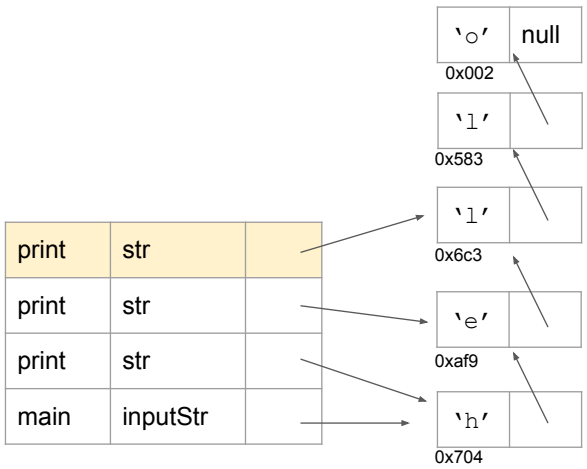
Output: `he`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

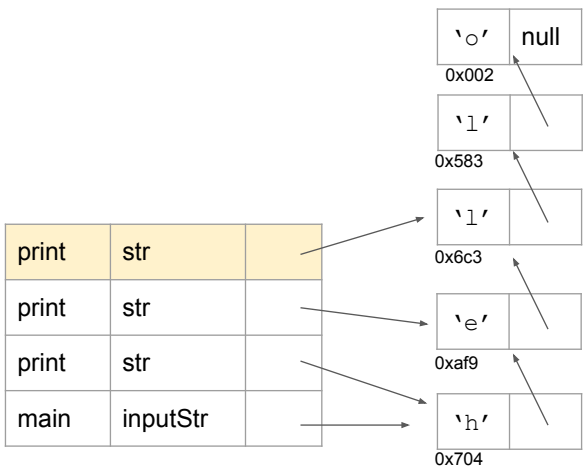
Output: `he`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

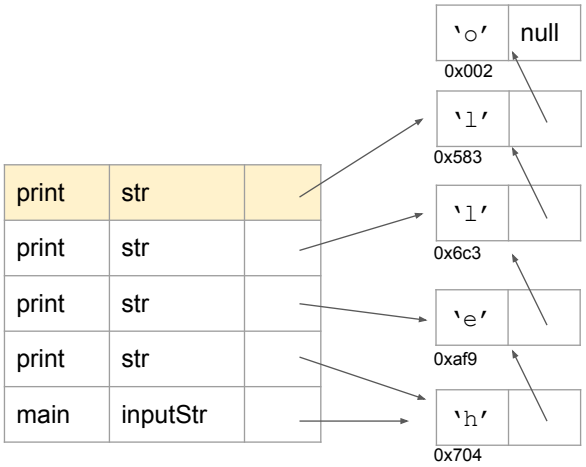
Output: `hel`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

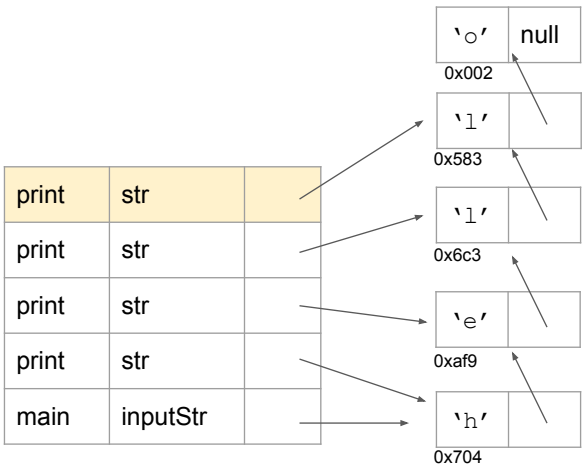
Output: `hel`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

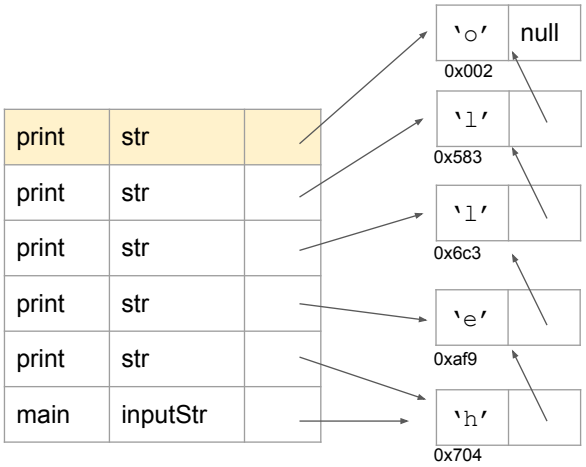
Output: `hell`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

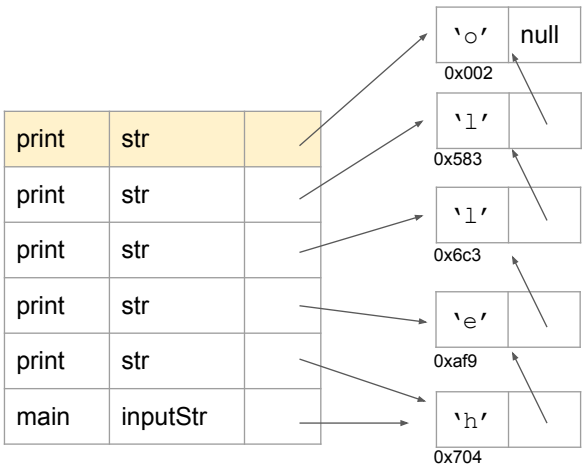
Output: `hell`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

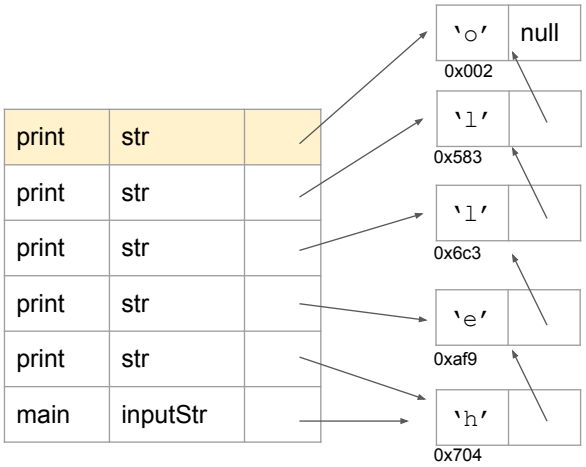
Output: `hello`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

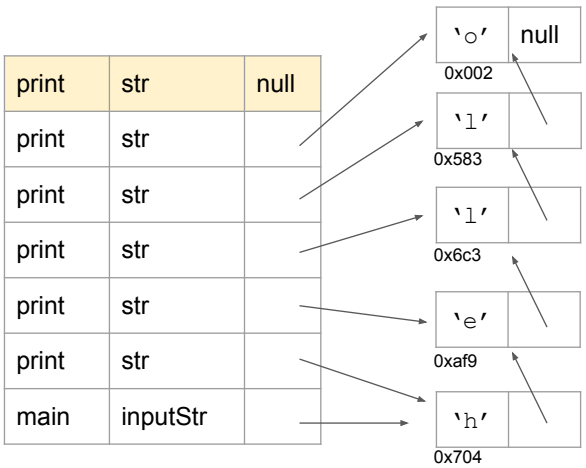
Output: `hello`



Tracing through `StringNode.print()`

```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

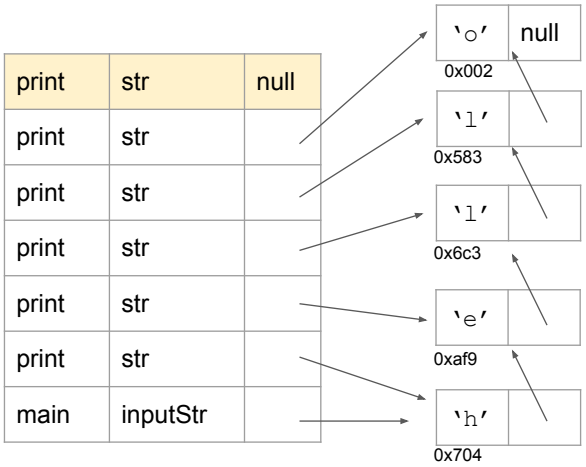
Output: `hello`



Tracing through `StringNode.print()`

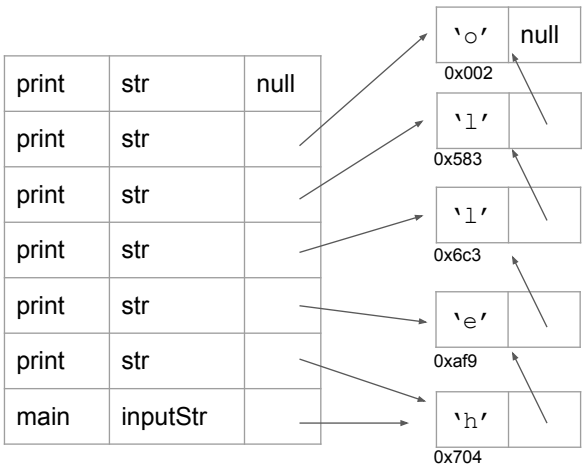
```
/*
 * Recursively process each node of
 * a linked list formed by
 * StringNodes, printing each one.
 */
public static void print(StringNode str)
{
    if (str == null) {
        return;
    } else {
        System.out.print(str.ch);
        print(str.next);
    }
}
```

Output: `hello`

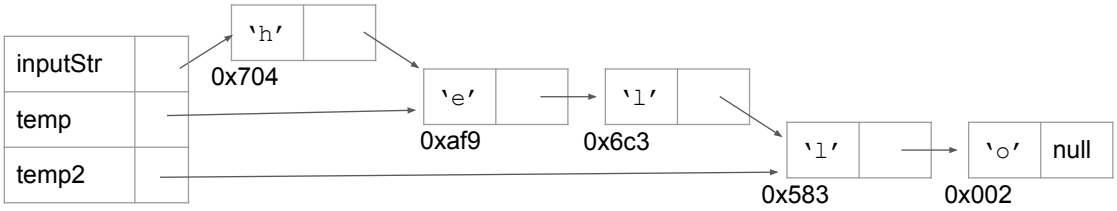


Tracing through `StringNode.print()`

Since this call hits the base case, we return, and pop each stack frame off of the stack, one by one, starting at the top.



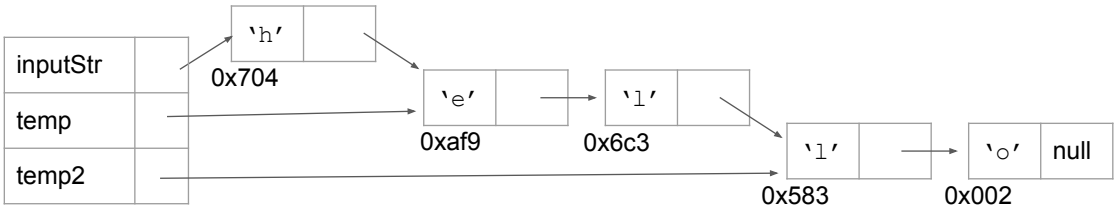
More practice with variable expressions and references



Specify how the following expressions evaluate:

- What is the value of `temp.next`?
- What does `temp.next.ch` evaluate to?
- What does `inputStr.next.next == temp` evaluate to?
- What are some ways we can access the character `'o'`?

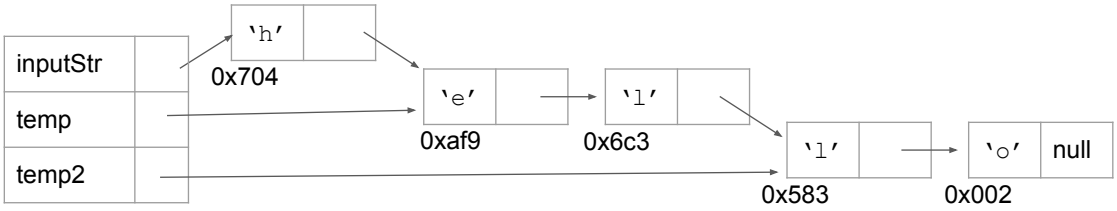
More practice with variable expressions and references



Specify how the following expressions evaluate:

- What is the value of `temp.next`? 0x6c3
- What does `temp.next.ch` evaluate to?
- What does `inputStr.next.next == temp` evaluate to?
- What are some ways we can access the character `'o'`?

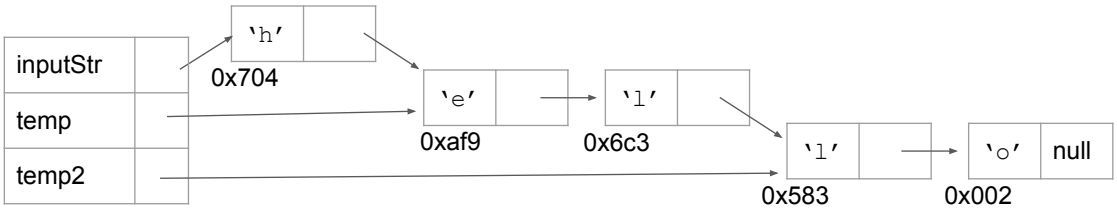
More practice with variable expressions and references



Specify how the following expressions evaluate:

- What is the value of `temp.next`? 0x6c3
- What does `temp.next.ch` evaluate to? The character `'l'`
- What does `inputStr.next.next == temp` evaluate to?
- What are some ways we can access the character `'o'`?

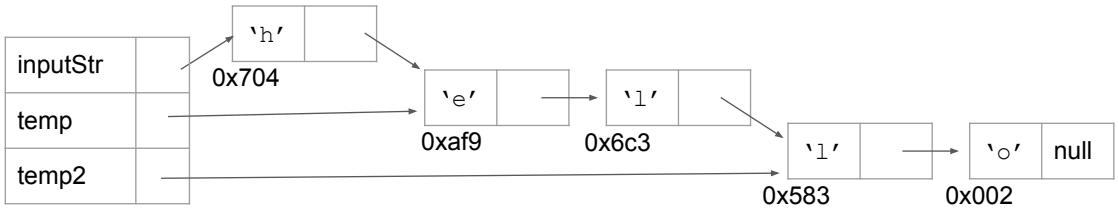
More practice with variable expressions and references



Specify how the following expressions evaluate:

- What is the value of `temp.next`? 0x6c3
- What does `temp.next.ch` evaluate to? The character `'l'`
- What does `inputStr.next.next == temp` evaluate to? false
- What are some ways we can access the character `'o'`?

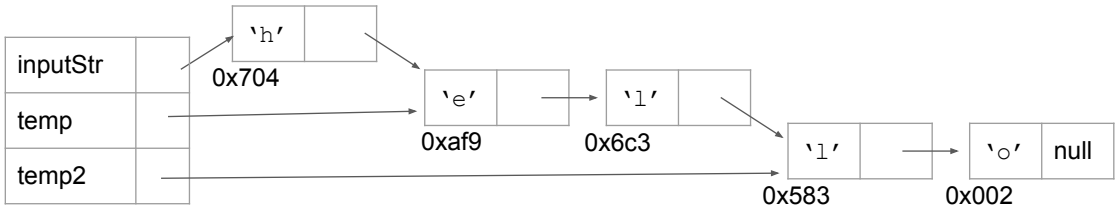
More practice with variable expressions and references



Specify how the following expressions evaluate:

- What is the value of `temp.next`? 0x6c3
- What does `temp.next.ch` evaluate to? The character `'l'`
- What does `inputStr.next.next == temp` evaluate to? false
- What are some ways we can access the character `'o'`? temp2.next.ch
  temp.next.next.next.ch
  inputStr.next.next.next.next.ch

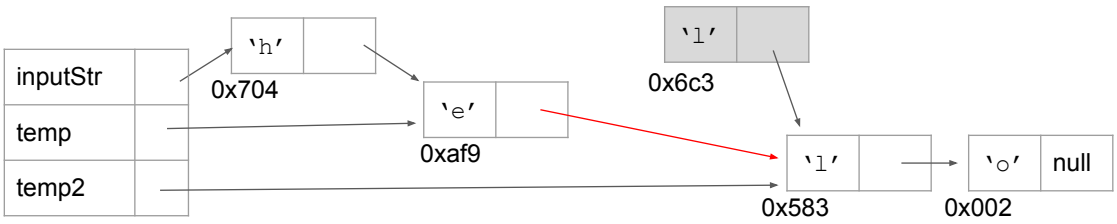
More practice with variable expressions and references



How do the following statements change the diagram?

- temp.next = temp2
- temp = temp2.next.next
- inputStr = inputStr.next
- temp2 = null

More practice with variable expressions and references

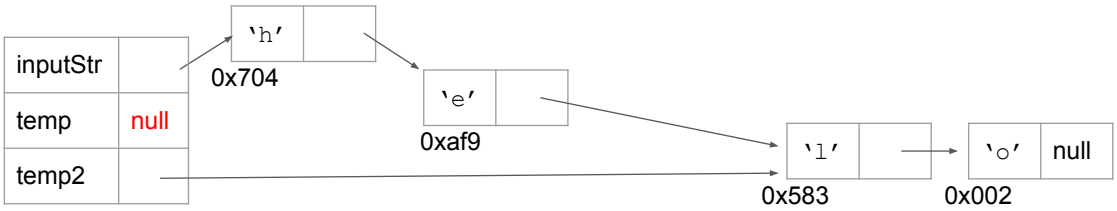


How do the following statements change the diagram?

- temp.next = temp2
- temp = temp2.next.next
- inputStr = inputStr.next
- temp2 = null

The node containing `'e'` now points to the second `'l'`-node, and we lose a reference to the first `'l'`-node.

More practice with variable expressions and references

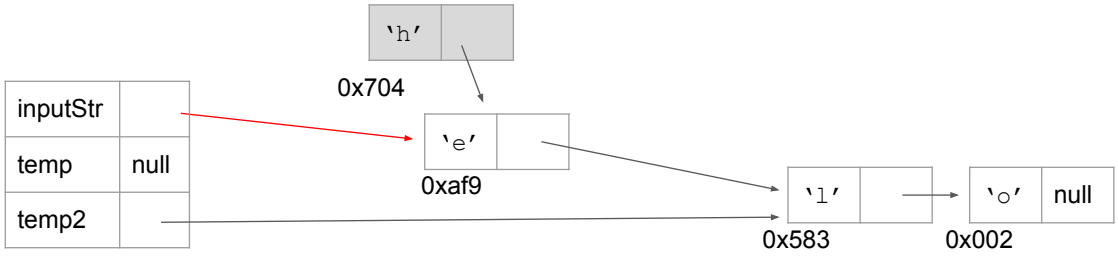


How do the following statements change the diagram?

- temp.next = temp2
- temp = temp2.next.next
- inputStr = inputStr.next
- temp2 = null

The `temp` variable now has `null`.

More practice with variable expressions and references

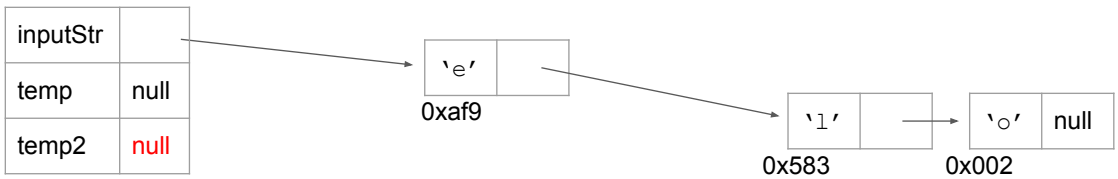


How do the following statements change the diagram?

- temp.next = temp2
- temp = temp2.next.next
- inputStr = inputStr.next
- temp2 = null

We lose a reference to the node at `0x704`, and `inputStr` now holds a reference to the node containing `'e'`.

More practice with variable expressions and references

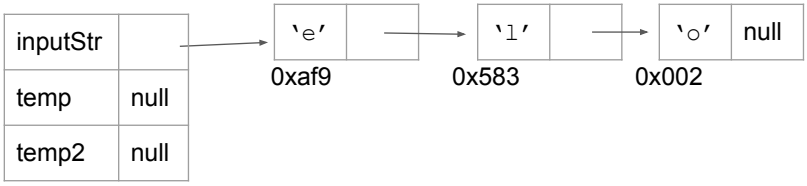


How do the following statements change the diagram?

- temp.next = temp2
- temp = temp2.next.next
- inputStr = inputStr.next
- temp2 = null

`temp2` now holds `null`

More practice with variable expressions and references



How do the following statements change the diagram?

- temp.next = temp2
- temp = temp2.next.next
- inputStr = inputStr.next
- temp2 = null

Converting a simple recursive method to an iterative one

```
/**
 * numOccurrences - find the number of
 * occurrences of the character ch in
 * the linked list to which str refers
 */
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;

    int occurInRest = numOccurrences(str.next, ch);

    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

This is the recursive version of the `numOccurrences()` method which finds the number of occurrences of a specified character in our linked list.

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str, char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

We want to convert this method into an iterative one. Think back to our trace through the `StringNode.print()` method. Do you see any similarities? What do you think the stack looks like when this method is executed? What does it return?

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str, char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

We want to convert this method into an iterative one. Think back to our trace through the `StringNode.print()` method. Do you see any similarities? What do you think the stack looks like when this method is executed? What does it return?

The stack will look almost identical — one recursive call per node, plus one for the base case when `str` holds a reference to `null`. Also, each stack frame will store a variable `occurInRest` that will keep the value of its recursive call, and will return either `occurInRest` or `occurInRest + 1`, depending on the reference held by `str` in that call.

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

Think about what we need to track in our iterative version. What variables will the iterative version need?

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

Think about what we need to track in our iterative version. What variables will the iterative version need?

We'll need a reference to the current `StringNode` we're looking at, and a counter for the number of occurrences.

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

We're going to need a loop. What kind of terminating condition should we have?

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

We're going to need a loop. What kind of terminating condition should we have?

It's probably a good idea to make the terminating condition somehow depend on reaching a `null` reference. Our extra variable that we keep updating will eventually get the value of `null` after we move it past the last node.

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

What will be in the body of our loop?

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

What will be in the body of our loop?

We'll need to look at the character in the current `StringNode`, and if the character is equal to the one we're counting, we'll need to increment the counter.

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

How will you advance through the list?

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    if (str == null)
        return 0;
    int occurInRest = numOccurrences(str.next, ch);
    if (str.ch == ch)
        return 1 + occurInRest;
    else
        return occurInRest;
}
```

How will you advance through the list?

We can use the `next` field of each `StringNode` to advance to the next `StringNode` in the list.

Converting a simple recursive method to an iterative one

```
public static int numOccurrences(StringNode str,  char ch) {
    int numOccur = 0;
    StringNode trav = str;

    while (trav != null) {
        if (trav.ch == ch)
            numOccur++;

        trav = trav.next;
     }

     return numOccur;
 }
```

Converting the `StringNode.read()` method

The read method takes an `InputStream` object, which can represent any input source which is processed as a sequence of bytes (such as the `System.in` input stream which allows us to read from the keyboard, or a file or network resource). The `IOException` isn't important; it's simply there in case the input stream goes down when the method is executed. `StringNode.read(System.in)` will read any string the user enters one character at a time and return a reference to a linked list which it constructs as the user types each character. Here is the original, recursive `read()` method:

```
/**
 * read - reads a string from an input stream and returns a
 * reference to a linked list containing the characters in the string
 */
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

If we're going to convert the `read()` method into an iterative one successfully, we should first trace what happens in the recursive method when it is invoked. If the `read()` method is called, and a character is fed into the stream, what happens?

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

If we're going to convert the `read()` method into an iterative one successfully, we should first trace what happens in the recursive method when it is invoked. If the `read()` method is called, and a character is fed into the stream, what happens?

The construction of a new node begins with the character from the stream in the `ch` field and the method is called recursively.

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

What does the method return? How does this fit into the creation of a new node?

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

What does the method return? How does this fit into the creation of a new node?

The method returns a reference to the StringNode it just created. When a recursive call to read() returns its reference to such a node, notice that it is inserted straight into the “next” field of the StringNode constructor in the previous stack frame!

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

Input string: "abcd"

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

Input string: "abcd"

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

Input string: "abcd"

Converting the `StringNode.read()` method

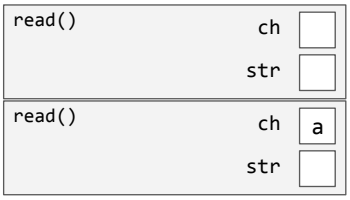
```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

Input string: "abcd"

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

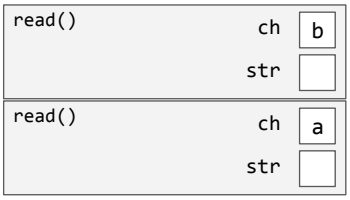
Input string: "abcd"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

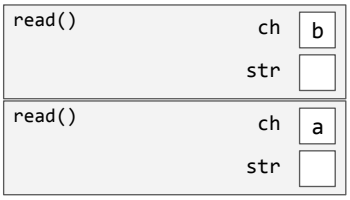
Input string: "abcd"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

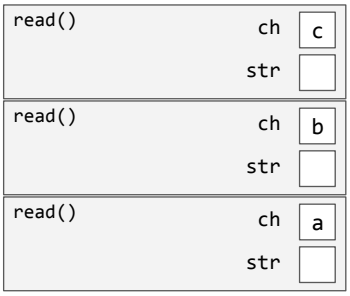
Input string: "abcd"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

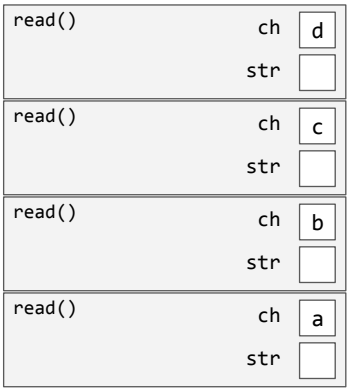
Input string: "abcd"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

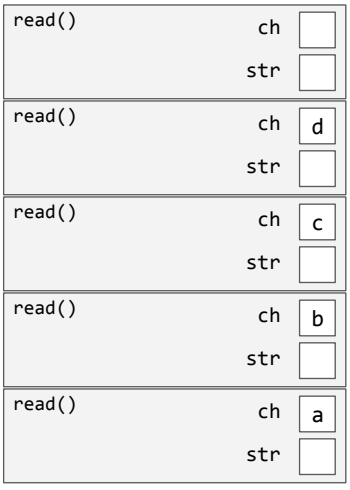
Input string: "abcd"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

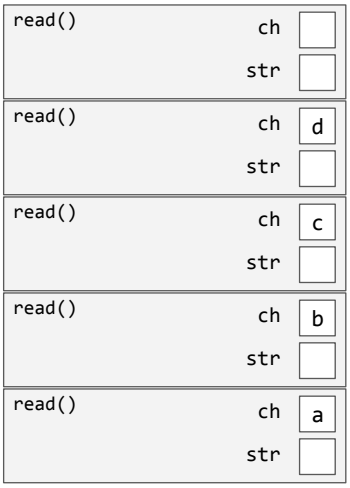
Input string: "abcd"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

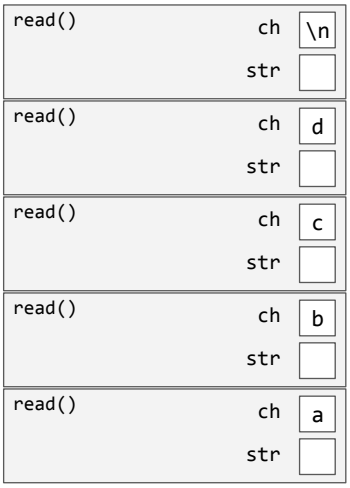
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

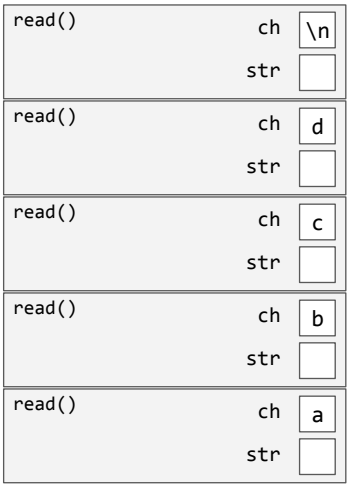
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

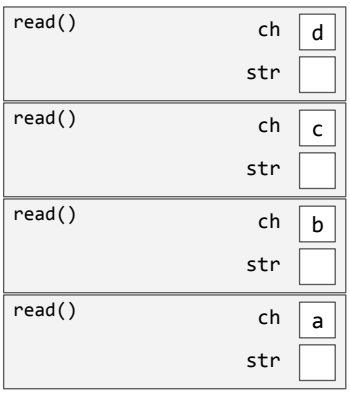
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

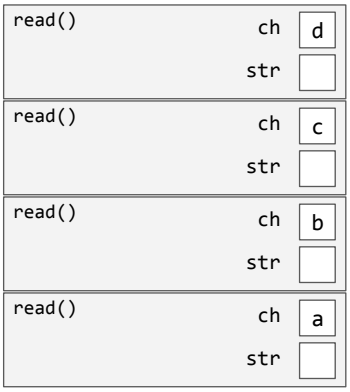
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else                                    null
        str = new StringNode(ch, read(in));

    return str;
  }
```

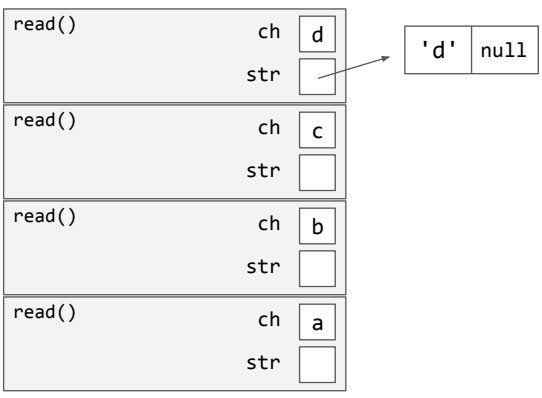
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else                                    null
        str = new StringNode(ch, read(in));

    return str;
  }
```

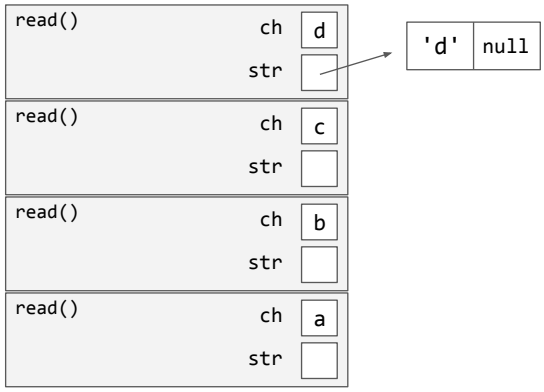
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
}
```

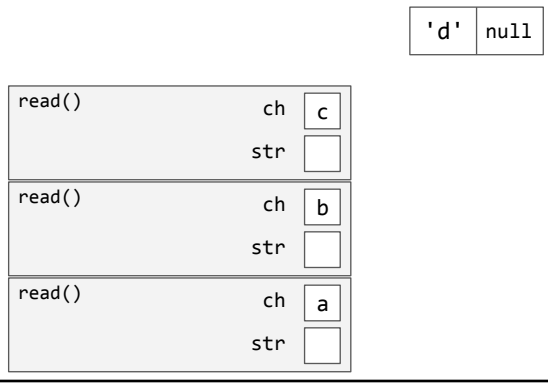
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
}
```

Input string: "abcd\n"

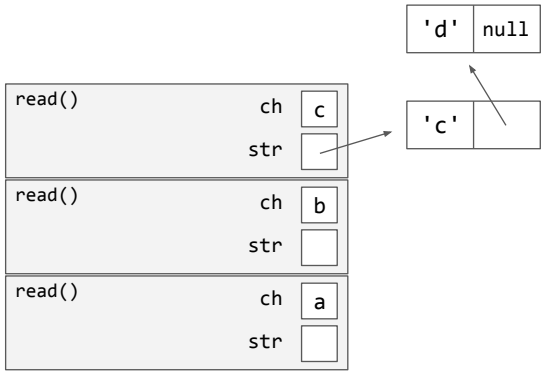


Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
}
```

reference to returned `StringNode`

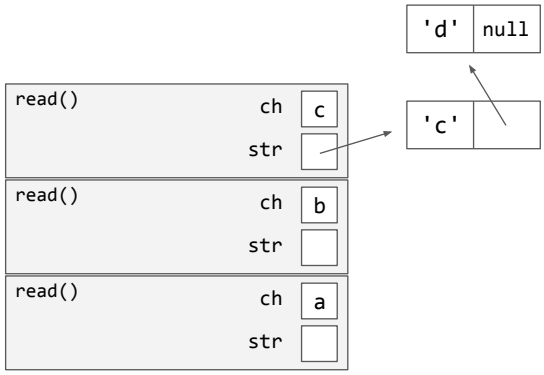
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
}
```

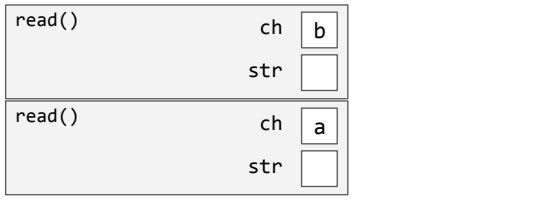
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

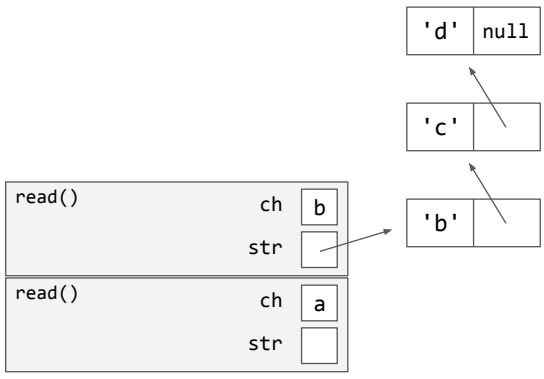
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

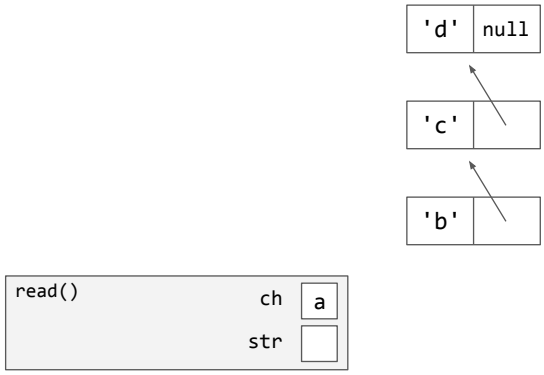
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

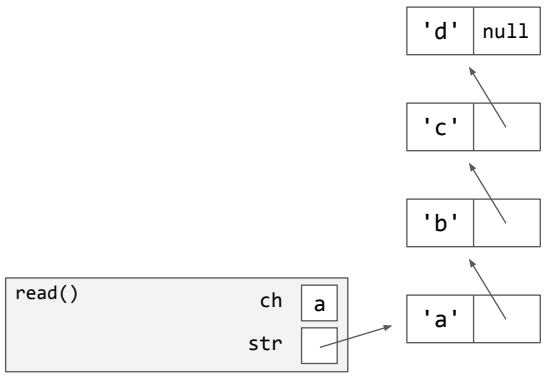
Input string: "abcd\n"



Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

Input string: "abcd\n"



Converting the `StringNode.read()` method

In what order does the `read()` method construct the linked list? Will this work as well in our iterative implementation? Why or why not?

Converting the `StringNode.read()` method

In what order does the `read()` method construct the linked list? Will this work as well in our iterative implementation? Why or why not?

The `read()` method creates a new stack frame for each character that is taken from the input stream, which contains the new `StringNode`. Once the base case is hit, the stack frames are popped off and the list is built from the back, in reverse order.

Converting the `StringNode.read()` method

In what order does the `read()` method construct the linked list? Will this work as well in our iterative implementation? Why or why not?

The `read()` method creates a new stack frame for each character that is taken from the input stream, which contains the new `StringNode`. Once the base case is hit, the stack frames are popped off and the list is built from the back, in reverse order.

If we inline the recursive tree of calls from before, we can see how the method constructs the new string:

```
return StringNode('a', StringNode('b', StringNode('c', StringNode('d', null))))
```

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
}
```

Clearly we'll need a different strategy in the iterative implementation, since we'll need to start at the front of the list if we want to keep the input in the order in which it is read from the input stream. How many pointers will we need if we want the iterative method to add new elements to the back and then return the reference to the first node of the list?

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

Clearly we'll need a different strategy in the iterative implementation, since we'll need to start at the front of the list if we want to keep the input in the order in which it is read from the input stream. How many pointers will we need if we want the iterative method to add new elements to the back and then return the reference to the first node of the list?

We'll need two, one that points to the beginning of our list, and one that points to the last node we added.

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

We will need a loop to iterate over the characters as they are read. What should the terminating condition for our loop be? (Think about the base case in the method above.)

Converting the `StringNode.read()` method

```
public static StringNode read(InputStream in) throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
  }
```

We will need a loop to iterate over the characters as they are read. What should the terminating condition for our loop be? (Think about the base case in the method above.)

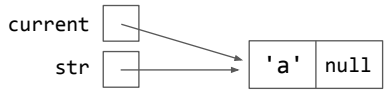
Our terminating condition should be a newline character.

```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();









  }
```

```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;









 }
```

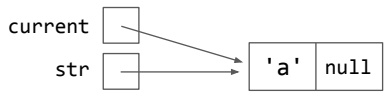
```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this









 }
```

```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this
```



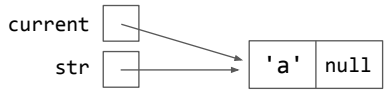

```
}
```

```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this
```

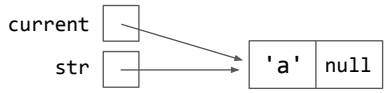



```
    return str;
}
```

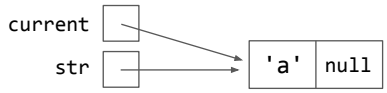
```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character




    return str;
 }
```



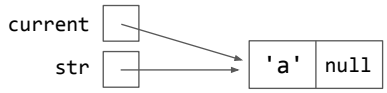

```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character
    while (           ) {




    }
    return str;
 }
```



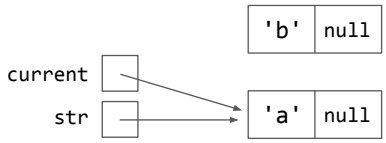
```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character
    while (ch != '\n') {




    }
    return str;
 }
```



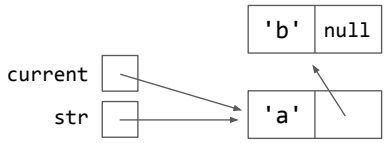

```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character
    while (ch != '\n') {
        current.next = new StringNode(ch, null);




    }
    return str;
 }
```



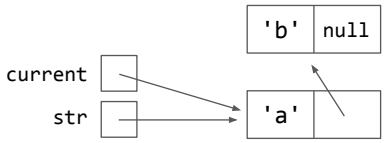
```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character
    while (ch != '\n') {
        current.next = new StringNode(ch, null);




    }
    return str;
 }
```



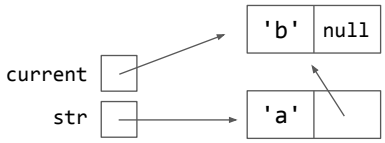

```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character
    while (ch != '\n') {
        current.next = new StringNode(ch, null);




    }
    return str;
 }
```



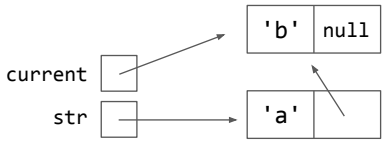
```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character
    while (ch != '\n') {
        current.next = new StringNode(ch, null);

        current = current.next;

    }
    return str;
 }
```




```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character
    while (ch != '\n') {
        current.next = new StringNode(ch, null);

        current = current.next;

    }
    return str;
 }
```



```
public static StringNode readIter(InputStream in)  throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);  // create the initial node
    str = current;                       // we will return this

    ch = (char)in.read()                 // read next character
    while (ch != '\n') {
        current.next = new StringNode(ch, null);

        current = current.next;
        ch = (char)in.read();
    }
    return str;
 }
```



Recursive Version:

```
public static StringNode read(InputStream in)
          throws IOException {
    StringNode str;
    char ch = (char)in.read();

    if (ch == '\n')     // base case
        str = null;
    else
        str = new StringNode(ch, read(in));

    return str;
 }
```

Why do you think this method was originally implemented recursively?

Iterative Version:

```
public static StringNode readIter(InputStream in)
          throws IOException {
    StringNode str = null;
    StringNode current;
    char ch = (char)in.read();

    if (ch == '\n')
        return str;

    current = new StringNode(ch, null);
    str = current;

    ch = (char)in.read()
    while (ch != '\n') {
        current.next = new StringNode(ch,
     null);

        current = current.next;
        ch = (char)in.read();
    }
     return str;
 }
```