

## How Can We Handle Long Strings? Fill Array...

### Caller to `stack_pop`

- **must provide a space** (an array) **for copy**.
- For safety, **must also pass length** of array.

### What should happen if caller passes an array shorter than the stored string?

- Fail? But their code pushed the string!
- Fill the array and add a NUL?
- Maybe the best choice in this case.

We will go with filling the array.

## A Function to Pop a String from a `stack_t`

```
// Returns 1 on success,
// or 0 on failure.
int32_t stack_pop (struct stack_t* s,
                  char* buf, int32_t len)
{
    int32_t i;
    char* read;
    if (stack_empty (s)) {
        return 0;
    }
}
```

Annotations:

- the stack (points to `struct stack_t* s`)
- the length (points to `int32_t len`)
- the array (points to `char* buf`)
- Stack is empty? Fail. (points to `if (stack_empty (s))`)
- for copying (points to the function body)

## Copy String into Buffer Provided by Caller

```
read = s->data[s->top];
for (i = 1;
     len > i && '\0' != *read;
     i++) {
    *buf++ = *read++;
}
```

Annotations:

- Copy from element on top of stack. (points to `s->data[s->top]`)
- Copy a character and advance pointers. (points to `*buf++ = *read++`)
- Loop (len - 1) times or until end of string. (points to the `for` loop)

## Finish the String, Pop the Element, and Return Success

```
*buf = '\0';
s->top++;
return 1;
}
```

Annotations:

- Write NUL to end of string. (points to `*buf = '\0'`)
- Pop copied element from stack. (points to `s->top++`)
- Pop has succeeded. (points to `return 1`)