Stacks

- Stacks are an abstract data type used primarily to organize data. They are most commonly implemented as either arrays or linked lists.

- Regardless of the underlying implementation, when data is added to the stack it sits “on top”, and so if an element needs to be removed, the most recently added element is the only one that can be.

- Last in, first out (LIFO).

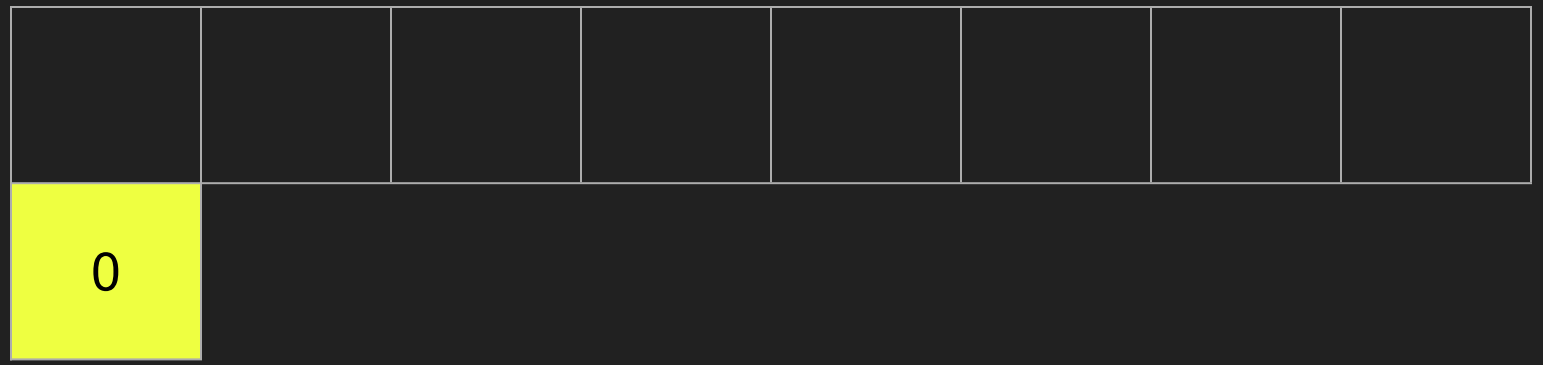
- Only two operations may be performed on stacks:
  - Push: Add a new element to the top of the stack.
  - Pop: Remove the most recently added element from the top of the stack.

- Stack implemented as an array:

```
typedef struct stack
{
    int array[CAPACITY];
    int top;
}
stack;
```

- Stack implemented as an array:

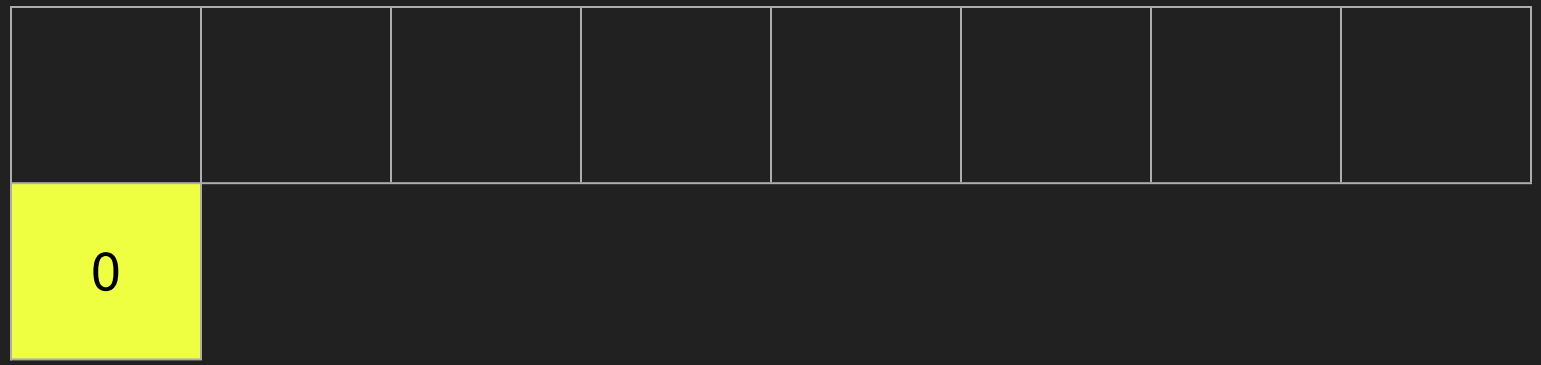
```
stack s;
s.top = 0;
```



- Push:
  - Accept a pointer to the stack (so that you can actually modify the contents notwithstanding push being a separate function).
  - Accept data to be added to the stack.
  - Add that data to the stack at the top of the stack.
  - Change the location of the top of the stack (so that the next piece of data can be properly inserted there.)

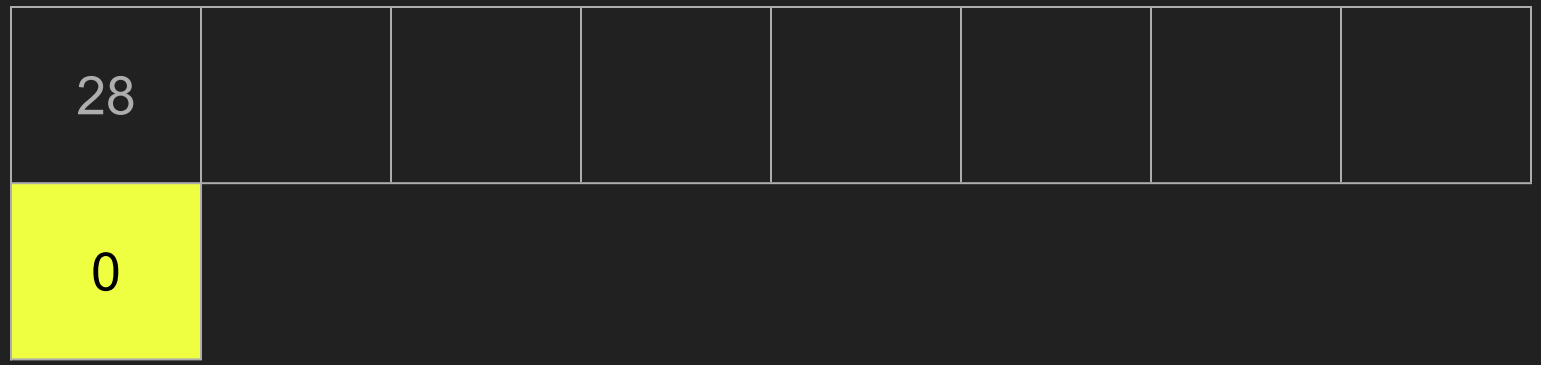
- Stack implemented as an array:

```
push(&s, 28);
```



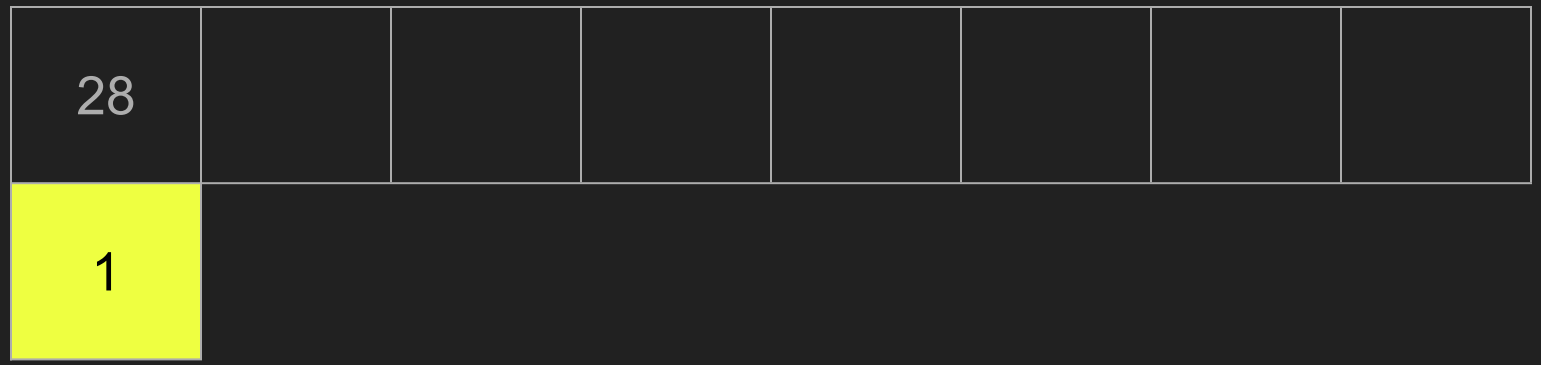
- Stack implemented as an array:

```
push(&s, 28);
```



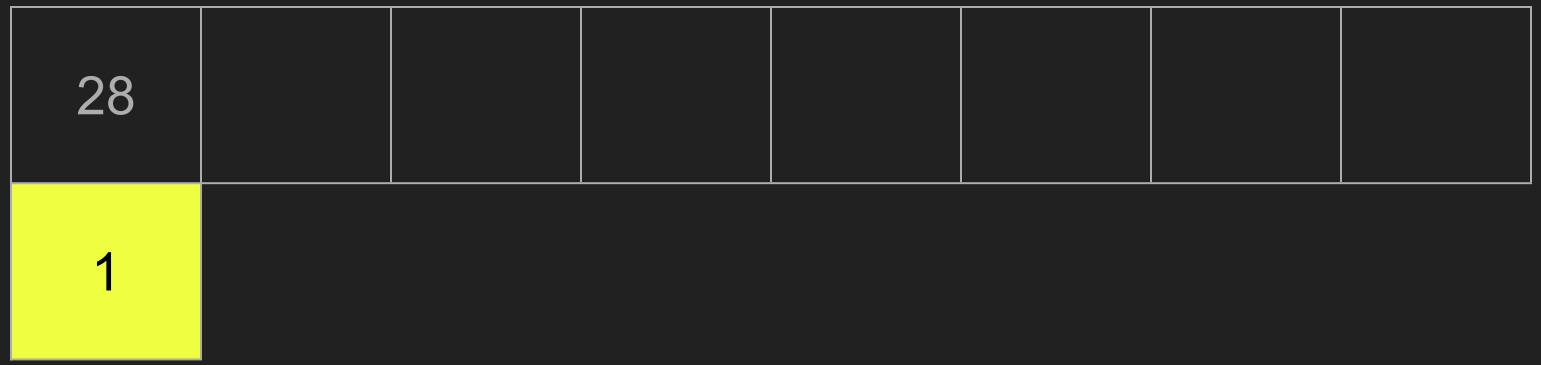
- Stack implemented as an array:

```
push(&s, 28);
```



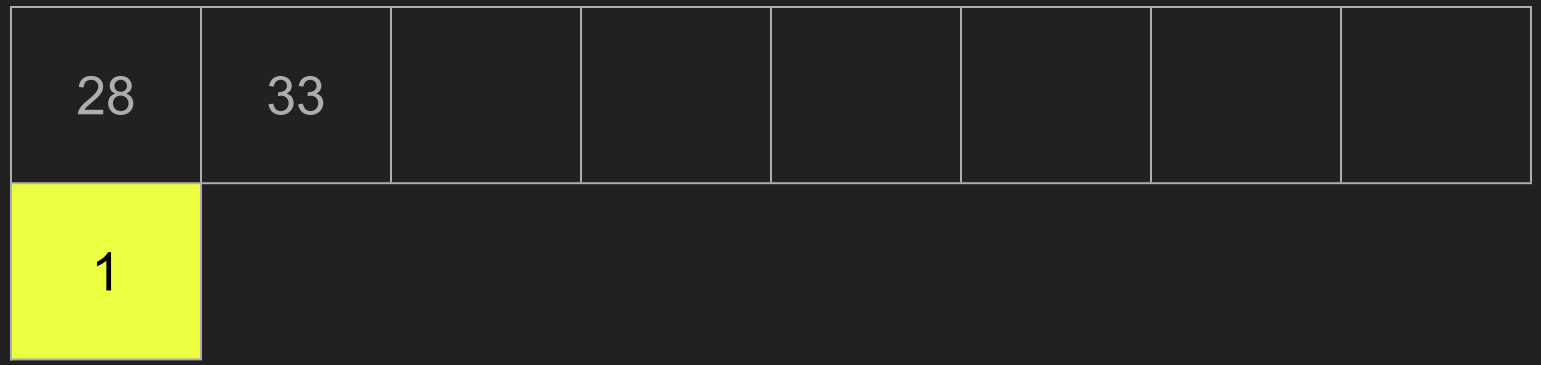
- Stack implemented as an array:

```
push(&s, 33);
```



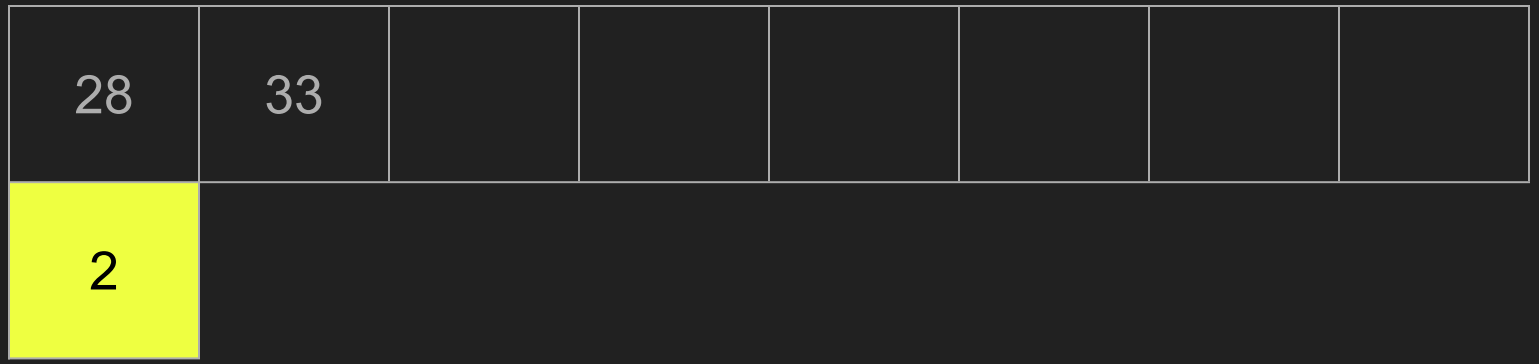
- Stack implemented as an array:

```
push(&s, 33);
```



- Stack implemented as an array:

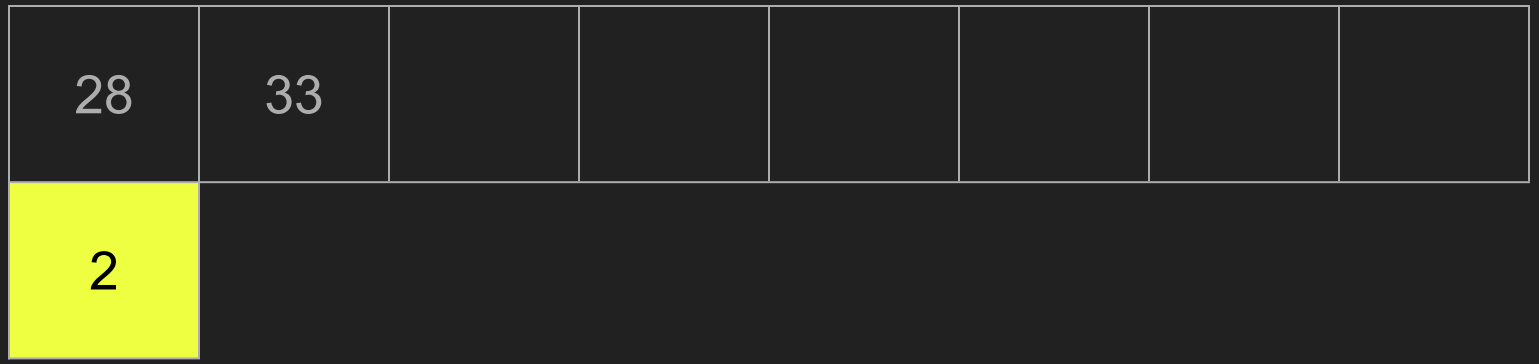
```
push(&s, 33);
```



- Pop:
  - Accept a pointer to the stack (so that you can actually modify the contents notwithstanding pop being a separate function).
  - Change the location of the top of the stack.
  - Return the value that was removed from the stack.

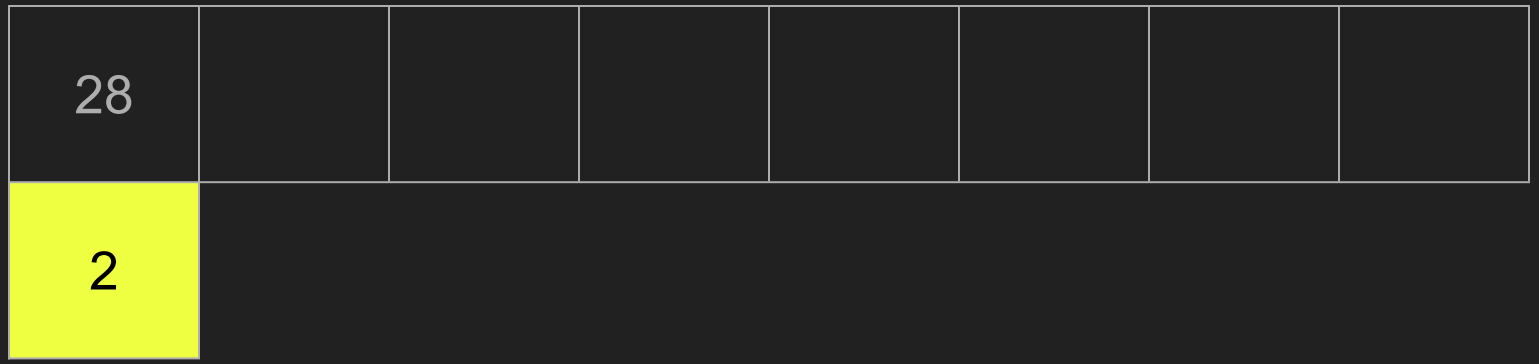
- Stack implemented as an array:

```
int x = pop(&s);
```



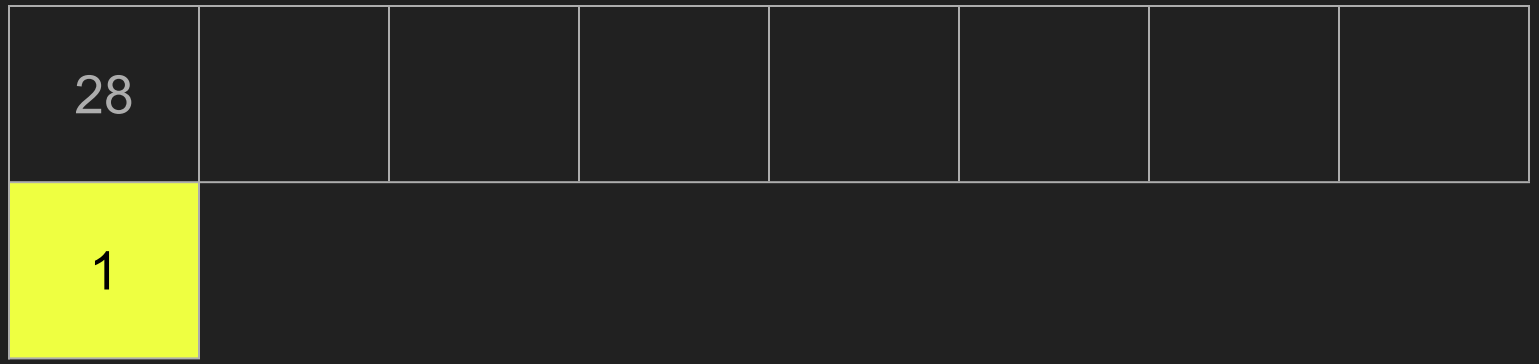
- Stack implemented as an array:

```
int x = pop(&s); // x gets 33
```



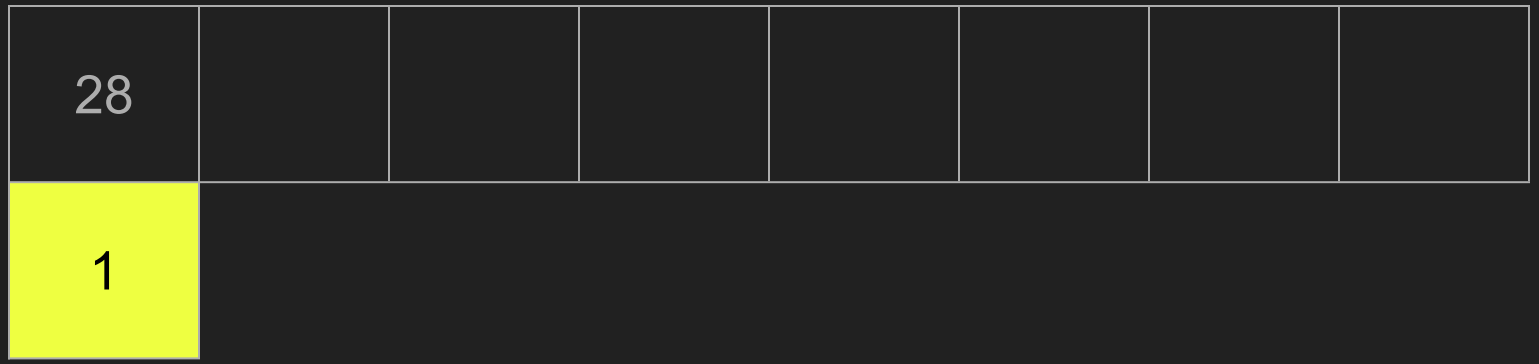
- Stack implemented as an array:

```
int x = pop(&s); // x gets 33
```



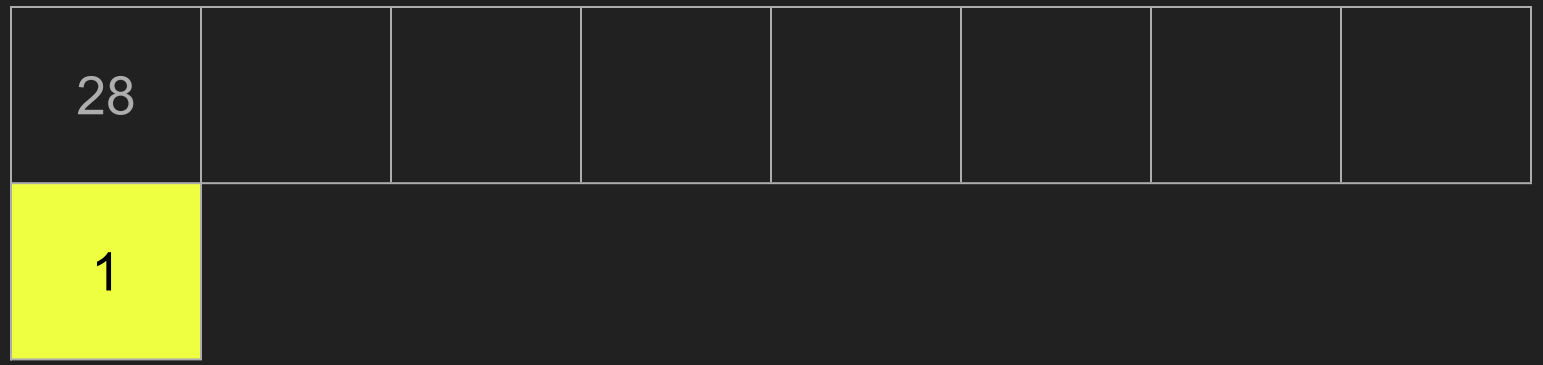
- Stack implemented as an array:

```
int x = pop(&s);
```



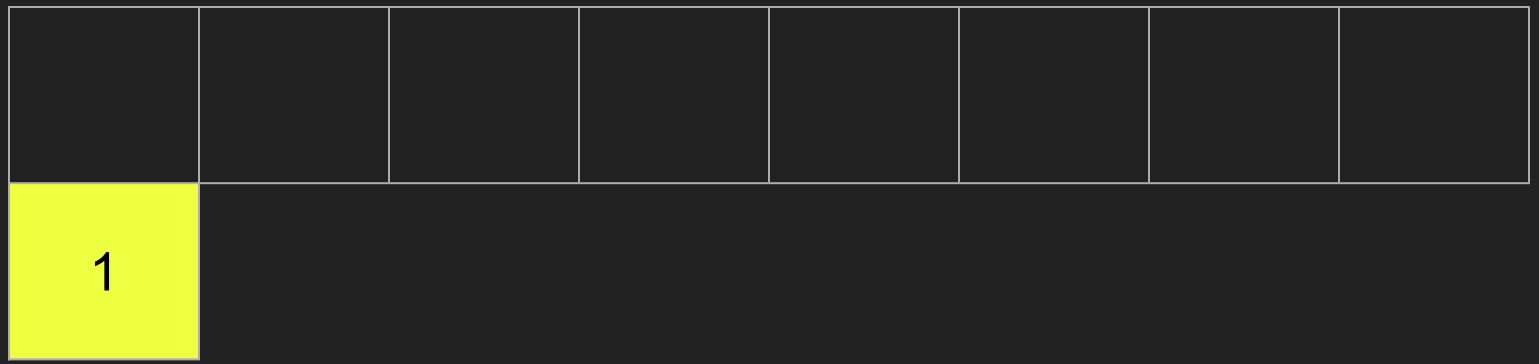
- Stack implemented as an array:

```
int x = pop(&s); // x gets 28
```



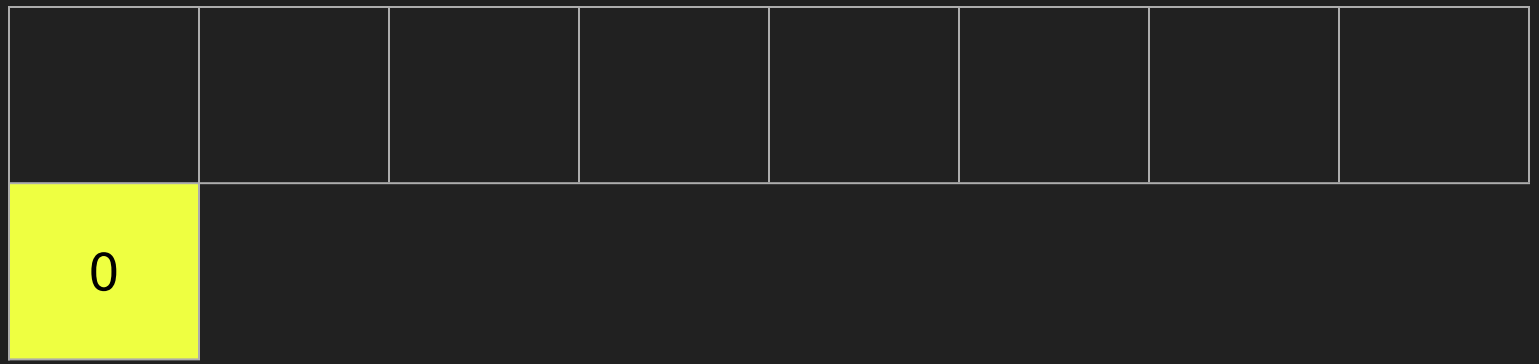
- Stack implemented as an array:

```
int x = pop(&s); // x gets 28
```



- Stack implemented as an array:

```
int x = pop(&s); // x gets 28
```



- Stack implemented as a linked list:

```
typedef struct stack
{
    int value;
    struct stack *next;
}
stack;
```

- Push:
  - Just like maintaining any other linked list.
  - Make space for a new list node, populate it.
  - Chain that node into the front of the linked list.
  - Make the new head of the linked list the node you just added.

- Pop:
  - Make the new head of the linked list the second node of the list (if you can).
  - Extract the data from the first node.
  - Free that node.