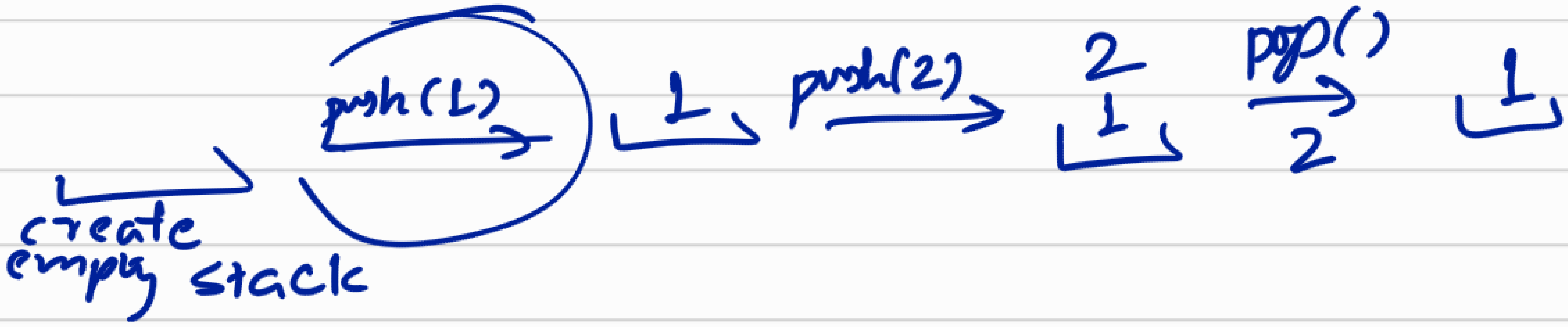
Data Structures :-

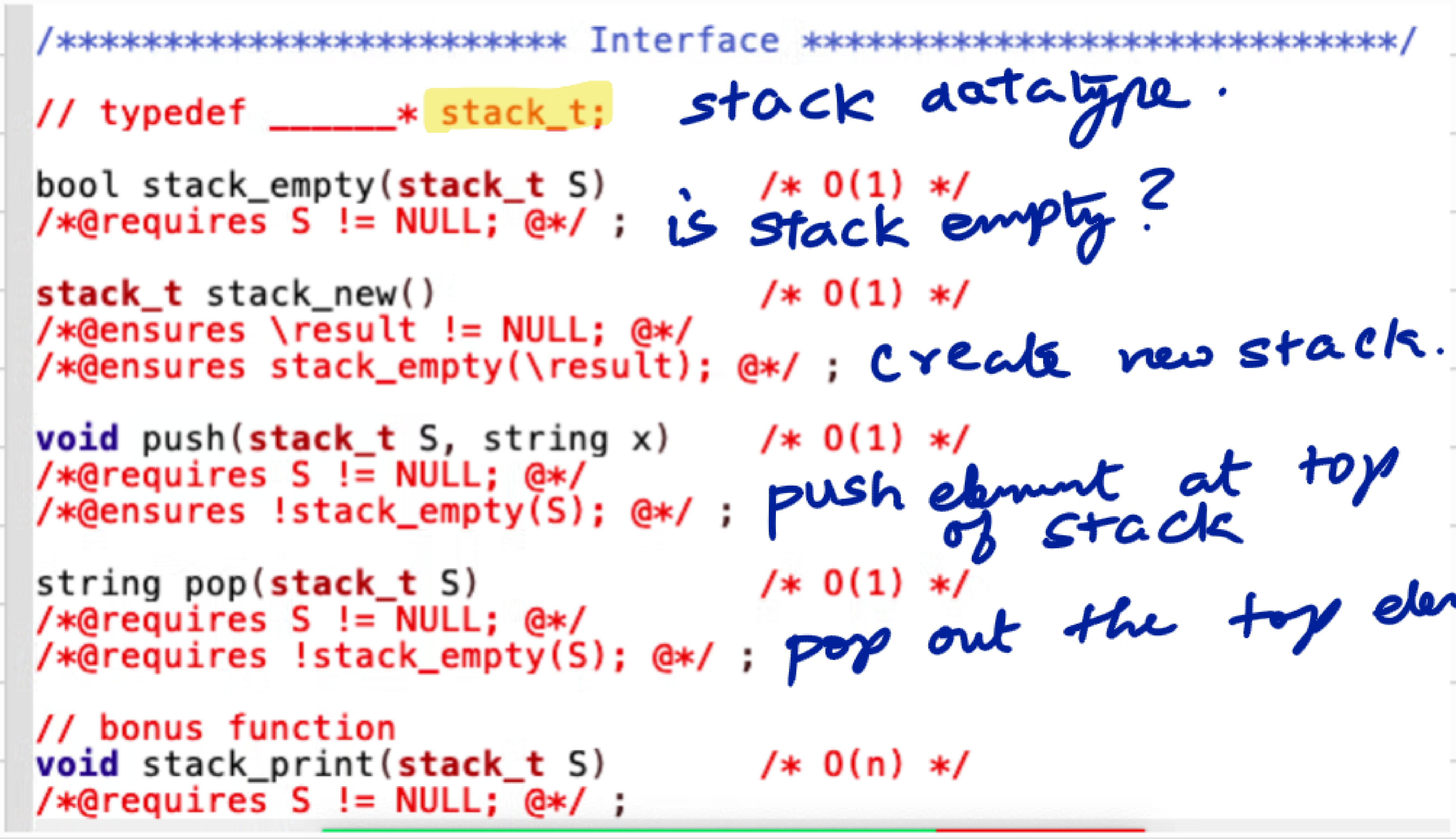
Two views to it :

Library side view ← developer
Client side view ← user.

Lists for organizing stuff :

Stacks : data structures with a LIFO (last in first out).





```
/*********************** Interface *****************************/

// typedef ______* stack_t;          stack datatype.

bool stack_empty(stack_t S)         /* O(1) */
/*@requires S != NULL; @*/ ;         is stack empty ?

stack_t stack_new()                 /* O(1) */
/*@ensures \result != NULL; @*/
/*@ensures stack_empty(\result); @*/ ;   create new stack.

void push(stack_t S, string x)      /* O(1) */
/*@requires S != NULL; @*/
/*@ensures !stack_empty(S); @*/ ;    push element at top of stack

string pop(stack_t S)               /* O(1) */
/*@requires S != NULL; @*/
/*@requires !stack_empty(S); @*/ ;   pop out the top element.

// bonus function
void stack_print(stack_t S)         /* O(n) */
/*@requires S != NULL; @*/ ;
```

Lets use this data structure from client side to organize reading list.

① You have a stack of books which you are reading.

② You pick the top book on the stack and read it, if the book is finished its taken off the stack (pop).

③ If you want to start a new book you put it at the top of the stack (push).

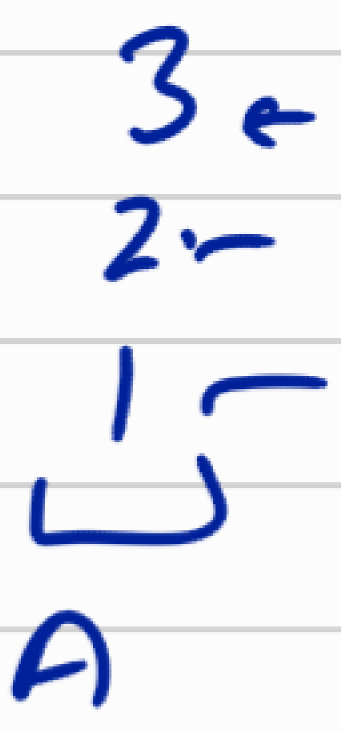
① Create a reading list of fiction. (stack_new)
② put book 1 at the top of reading list (push)
③ When finished with a book strike it off; take it off the stack. (pop ?.
④. how to know if nothing remains to be read (stack-empty ?.

```
stack-t rl = stack- new();
  rl. push ( 1) ;
  rl. push (2) ;
  rl. pop ( ) ;
        .
        .
        .
        .
        .
  _________;
```

What are you reading currently?

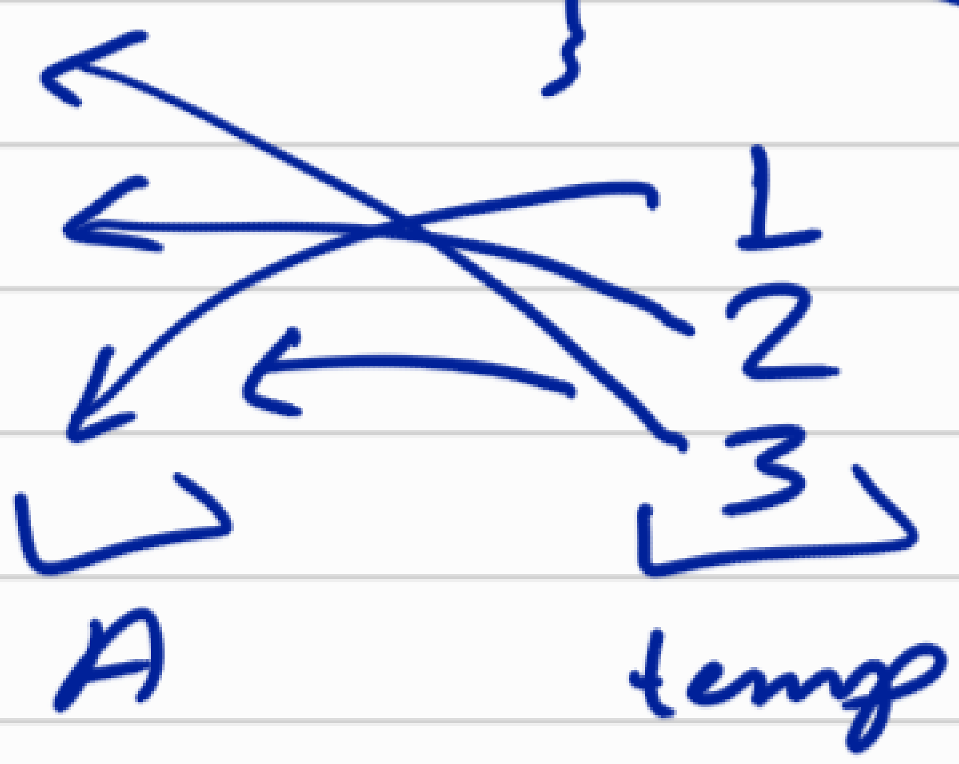
```
string peek ( stack-t A) {
   if ( stack-empty (A) ){ print ("empty stack");
 string current = pop( A);        return "";  }
  print ( current);
    push(A, current );
}
```

how many elements does A have?



```
int size( stack_t A) {
    int count = 0;
    while(! stack_empty( A)) {
        string t= pop (A) ;   push (temp,t);
        count ++;
    }
    return count;
}
```

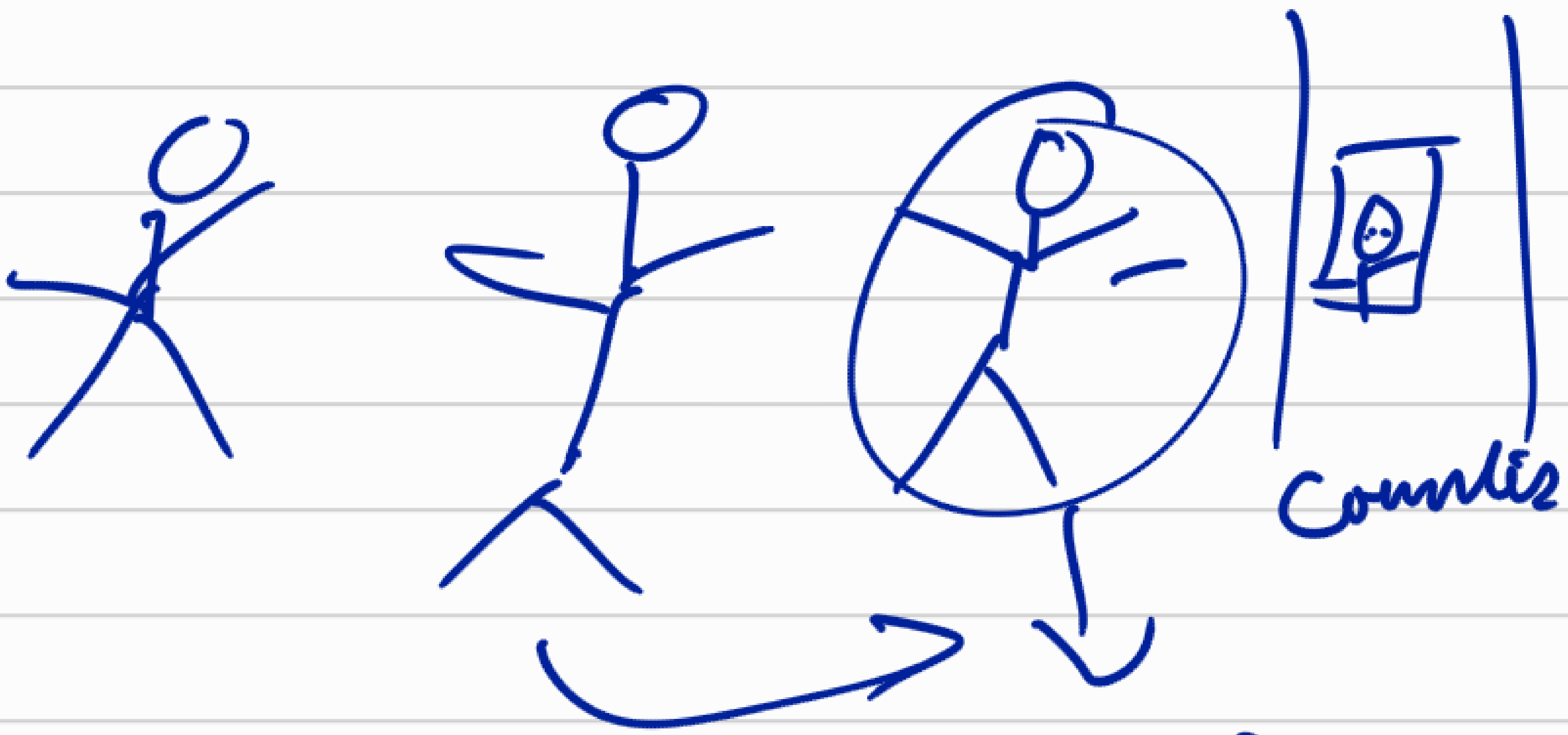
this destroys our stack A!!



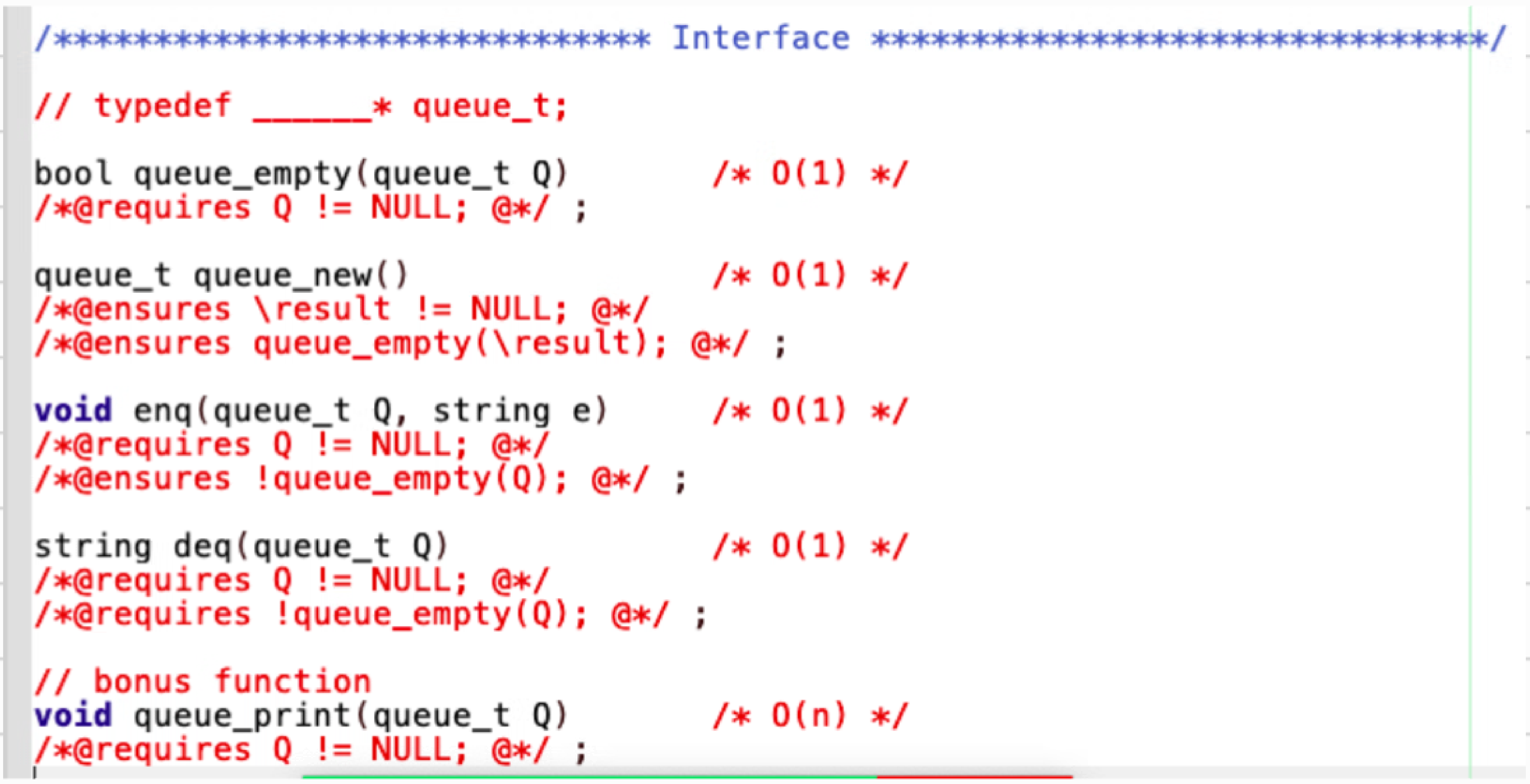
full Algo

```
count = 0
temp is a new empty stack.
till A is not empty :
        push(temp, pop(A))
        count ++
till temp is not empty :
        push(A, pop(temp))
return count
```

Queues: for ex. railway queue.

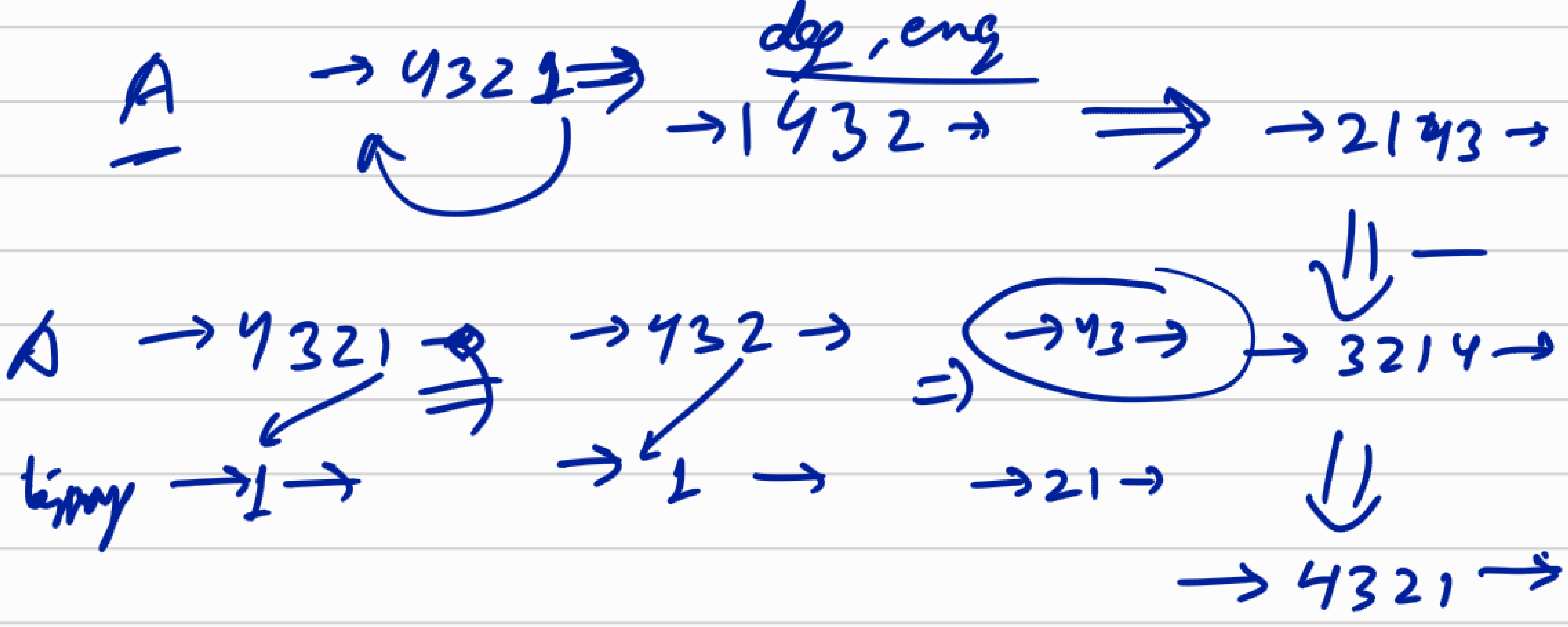


FIFO (first in first out)



```
/******************************** Interface *********************************/

// typedef ______* queue_t;

bool queue_empty(queue_t Q)          /* O(1) */
/*@requires Q != NULL; @*/ ;

queue_t queue_new()                  /* O(1) */
/*@ensures \result != NULL; @*/
/*@ensures queue_empty(\result); @*/ ;

void enq(queue_t Q, string e)        /* O(1) */
/*@requires Q != NULL; @*/
/*@ensures !queue_empty(Q); @*/ ;

string deq(queue_t Q)                /* O(1) */
/*@requires Q != NULL; @*/
/*@requires !queue_empty(Q); @*/ ;

// bonus function
void queue_print(queue_t Q)          /* O(n) */
/*@requires Q != NULL; @*/ ;
```

Find size of the queue.



Same idea as that of stacks :-

exchange :-

push ↔ enq

pop ↔ deq

stack ↔ queue