

Reading: GT 2.0, p. 56, GT 2.1.1–2.1.2, pp. 57–64, GT 1.3.3, pp.24–27

The most important concepts of this lecture are the abstract data types of *stack* and *queue* (i.e. what operations are expected of stacks and queues and what is the semantics of these operations).

In 1.3.3 focus on the concept of *loop invariant*.

Ingredients: Stacks, Stacks in PLs,

## The Stack Data Structure

Operations:

PUSH( $o$ ) – insert object  $o$  at the top

POP() – remove and return the top element (error if empty)

Stack semantics is LIFO = *last in first out*

## An Array-based Implementation

An  $n$ -element array  $S$  and an integer index  $t$  that points to the top of the stack.

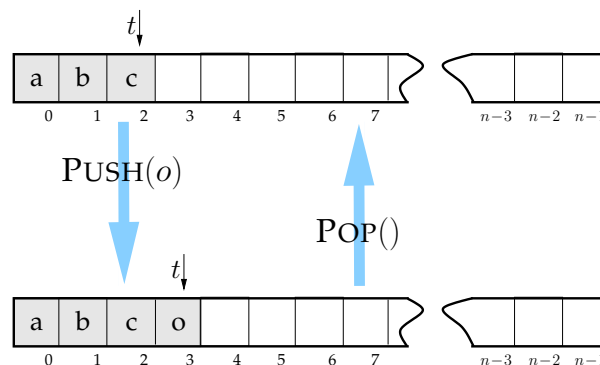


Figure 1: An array-based stack, before and after PUSHing an object  $o$ .

PUSH( $o$ )

```

1  if  $t = n - 1$ 
2      then error
3   $t \leftarrow t + 1$ 
4   $S[t] \leftarrow o$ 

```

POP()

```

1  if  $t = -1$ 
2      then error
3   $e \leftarrow S[t]$ 
4   $S[t] \leftarrow \text{NULL}$ 
5   $t \leftarrow t - 1$ 
6  return  $e$ 

```

## Stacks in Implementation of Programming Languages

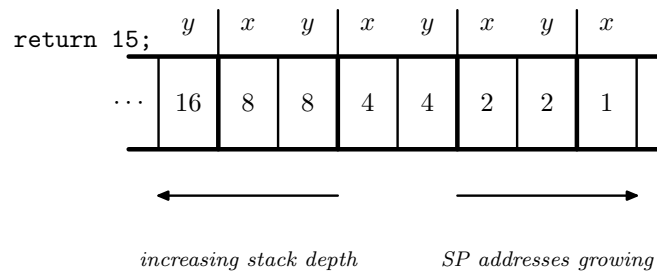
$F(x : \text{int}) : \text{int}$

```

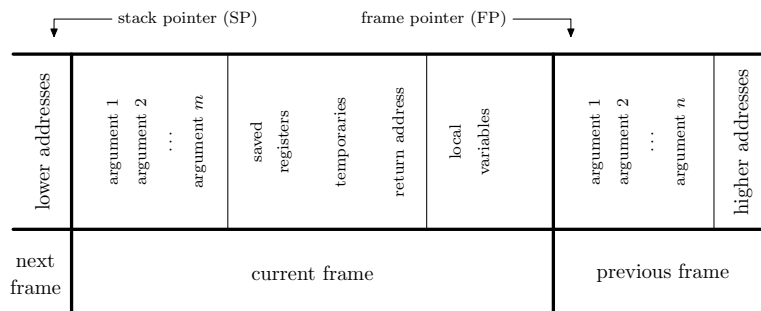
1   $y \leftarrow 2x$ 
2  if  $2x < 10$ 
3      then return  $F(y)$ 
4      else return  $y - 1$ 

```

The (abstract) stack during the evaluation of  $F(1)$ :



There is a lot more on the actual stack. Here is a more detailed view:



In short: the same constant space for all calls of a specific function, only depends on the parameters, local variables and the return type.

**Remember: Function calls (including recursion) cost time and space!**

## The Queue Data Structure

Operations:

ENQUEUE( $o$ ) – insert object  $o$  at the rear

DEQUEUE() – remove and return the object at the front (error if empty)

Queue semantics is FIFO = *first in first out*

## An Array-based Implementation

An array  $Q$  of  $n$  cells and two integer counters  $f$  (front) and  $r$  (rear).

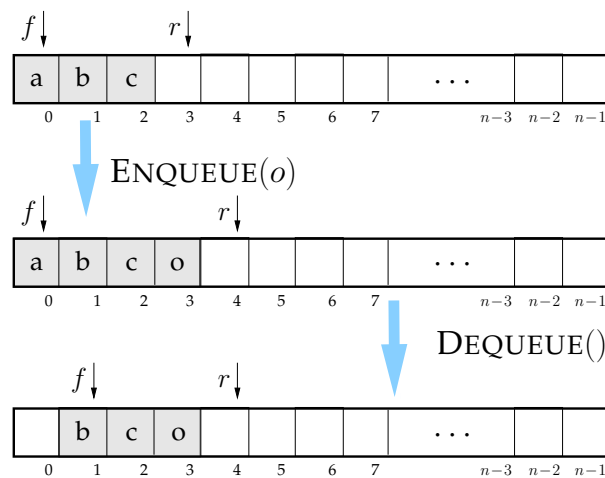


Figure 2: Invariant: the array is empty iff  $f = r$ .

Incrementation:  $f \leftarrow (f+1) \bmod n$  on dequeue,  $r \leftarrow (r+1) \bmod n$  on enqueue

Also called a *ring buffer*, an implementation often used in low-level programming (for example embedded systems). Fast, requires no dynamic memory management, but note a built-in limitation for the queue size.