Lecture 3

Stack

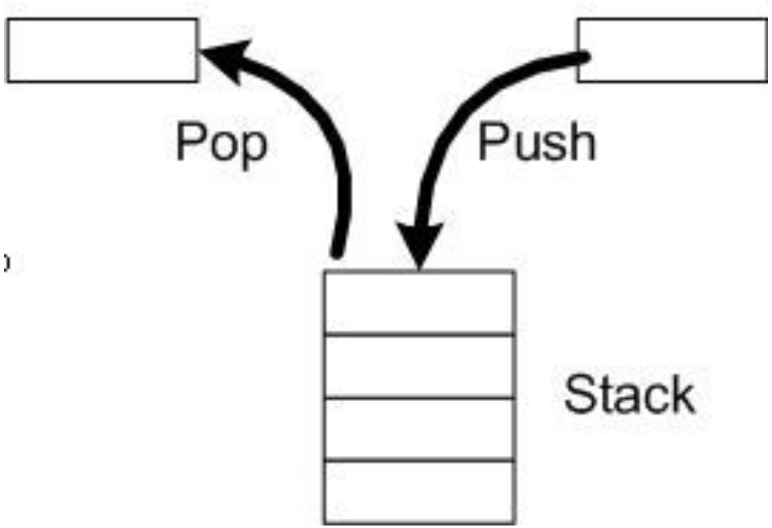

Stack

The Stack ADT (Abstract Data Type)

A **Stack** is a collection of objects inserted and removed according to the Last In First Out (LIFO) principle. Think of a stack of dishes.

Stack operations

Push and **Pop** are the two main operations



- When using push() operation to place the following items on a stack:

push(10)

push(20)

push(30)

push(0)

push(-30)

the output when popping from the stack is:

A: 10, 20, 30, 0 , -30

B: -30, 0, 10, 20, 30

C: 30, 10, 20, 0, -30

D: -30, 0, 30, 20, 10

E: 0, 30, -30, 10, 20

A lot of applications

- Think of the undo operation of an editor. The recent changes are pushed into a stack, and the undo operation pops it from the stack.
- Reverse strings
- The expression evaluation stacks are also used for parameter passing and local variable storage.
    - Think of ED diagrams and recursions!
- Check if a given expression has correct "(" , ")" order.

Classic Example: Parenthesis checker

(2 + 3) - (4 + 1)

- Push “(“
- Ignore 2,“+”, 3
- If you see”)” then Pop “(”. Exists?
- Ignore “-”
- Push “(“
- Ignore 4, “+”, 1
- If you see”)” then Pop “(”. Exists?
- Empty Stack, empty Input! Hooray!

Implementation. Arrays

Main update methods:

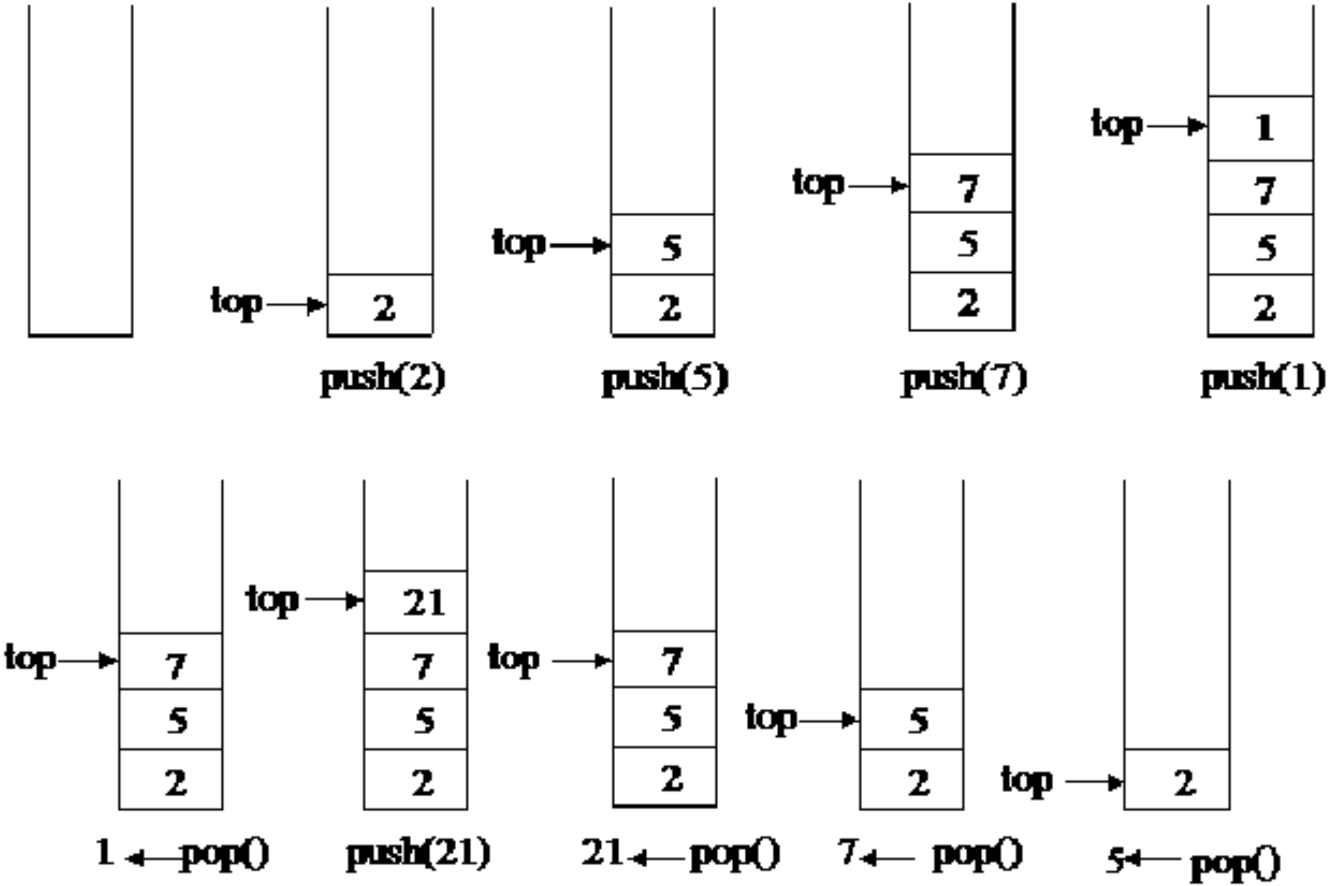
- Push(e): add an element to the stack
- Pop( ): remove an element from the stack

Additional useful methods

- Peek(): Same as pop, but does not remove the element
- Empty(): Boolean, True when the stack is *empty*
- Size(): Returns the size of the stack

REMINDER

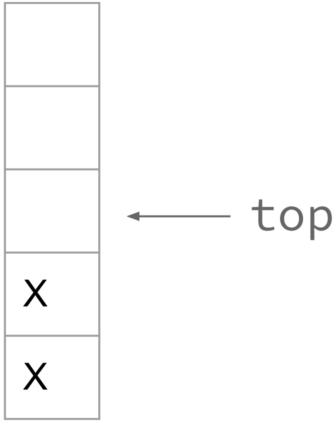
- `mic`


top
2
push(2)
top
5
2
push(5)
top
7
5
2
push(7)
top
1
7
5
2
push(1)
top
7
5
2
1 ← pop()
top
21
7
5
2
push(21)
top
7
5
2
21 ← pop()
top
5
2
7 ← pop()
top
2
5 ← pop()

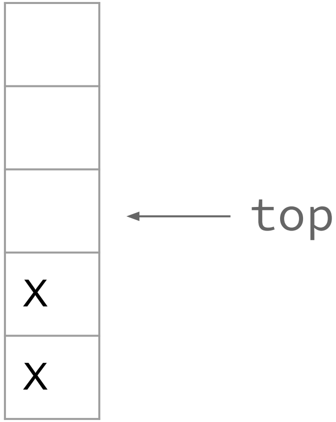
Implementation design

x
x

Implementation design

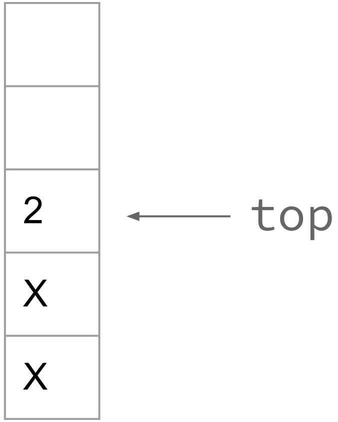


Implementation design



push(2)

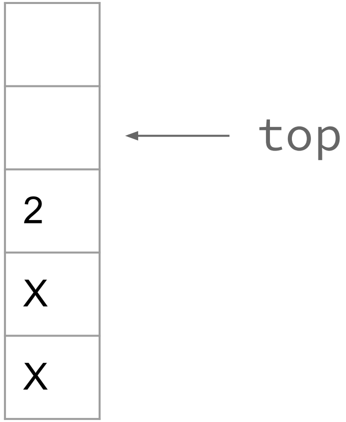
Implementation design



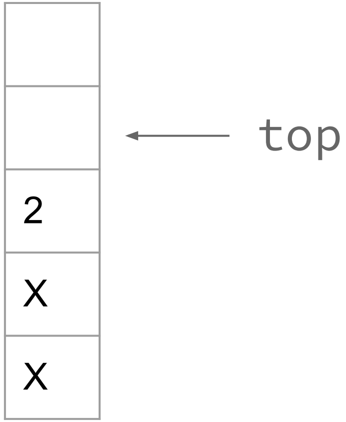
push(2)

Implementation design

push(2)

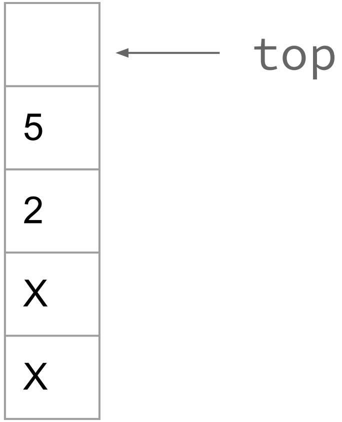


Implementation design



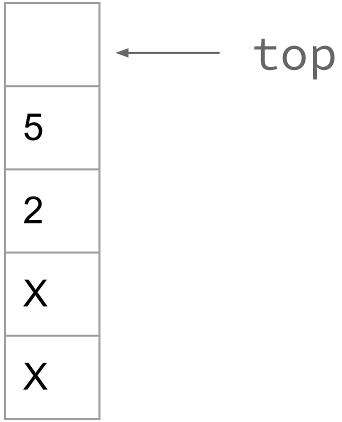

```
push(2)
push(5)
```

Implementation design



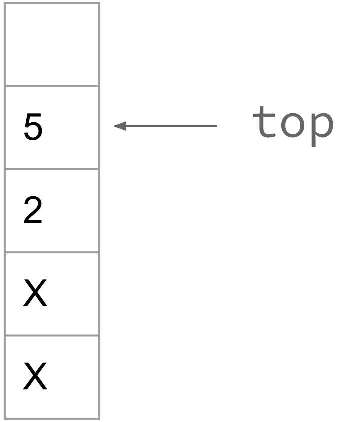

```
push(2)
push(5)
```

Implementation design



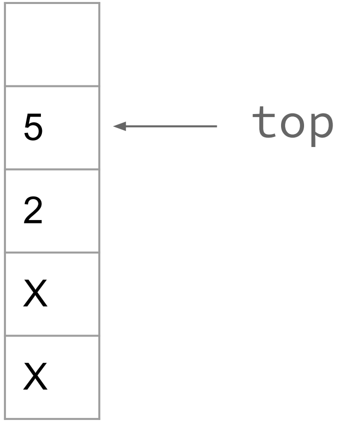

```
push(2)
push(5)
pop()
```

Implementation design



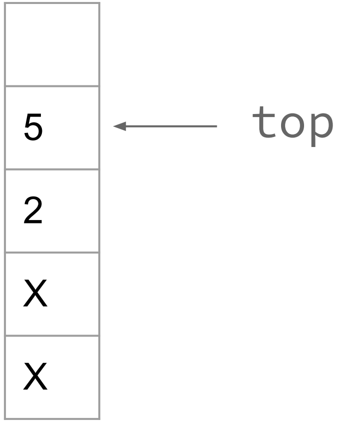

```
push(2)
push(5)
pop()
```

Implementation design



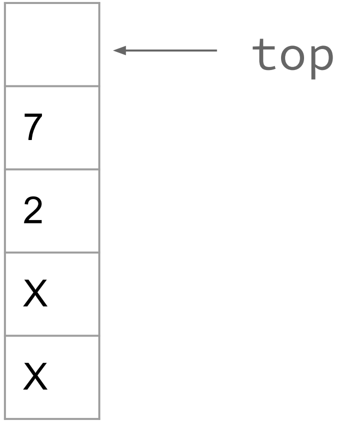

```
push(2)
push(5)
pop() //5 is returned
```

Implementation design



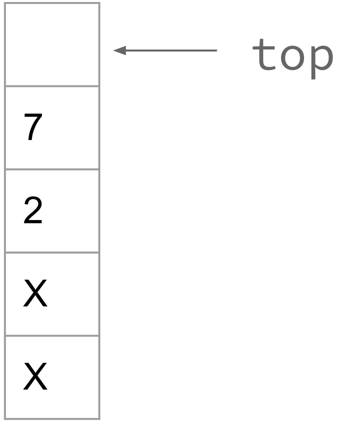

```
push(2)
push(5)
pop() //5 is returned
push(7)
```

Implementation design



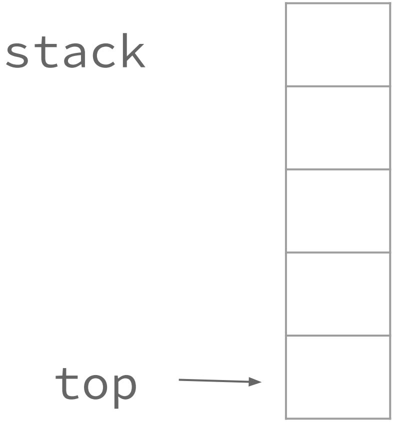

```
push(2)
push(5)
pop() //5 is returned
push(7)
```

Implementation design



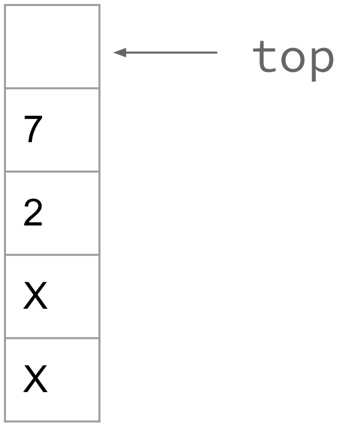

```
push(2)
push(5)
pop() //5 is returned
push(7)
```

```
push (int elem) {
     stack[top]= elem;
     top++;
}
```

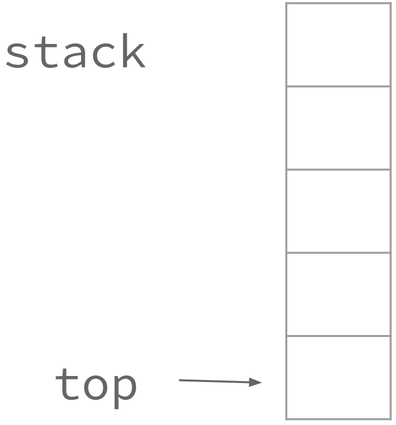


top = 0

Implementation design




```
push(2)
push(5)
pop() //5 is returned
push(7)
```




```
top = 0
```

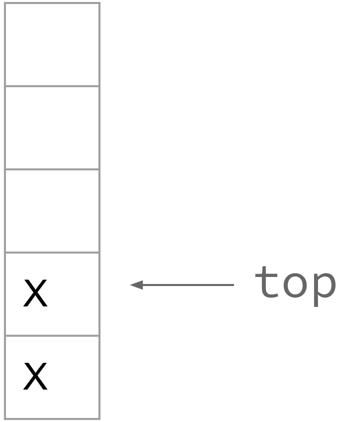
```
push (int elem) {
     stack[top]= elem;
     top++;
}
```

```
int pop() {
 top --;
 return stack[top];
}
```

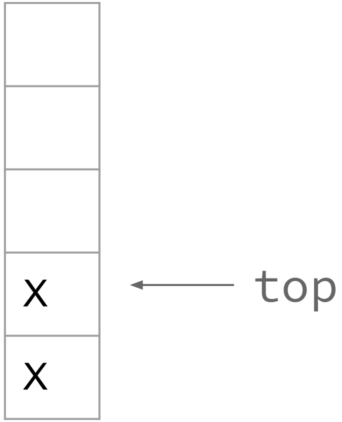
Implementation design

x
x

Implementation design

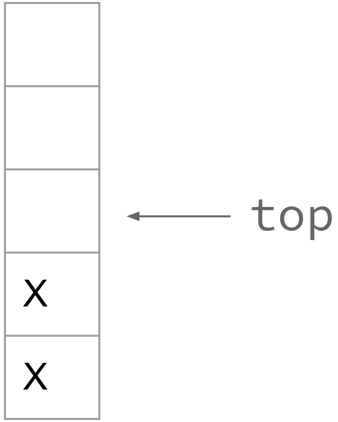


Implementation design



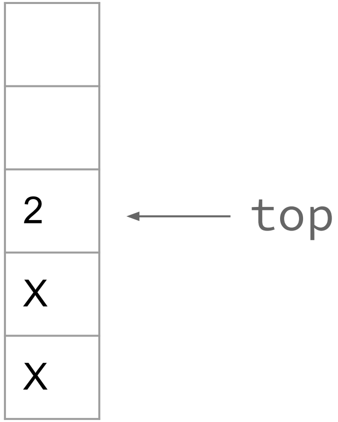
push(2)

Implementation design



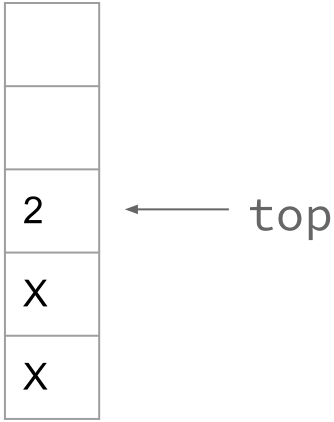
push(2)

Implementation design



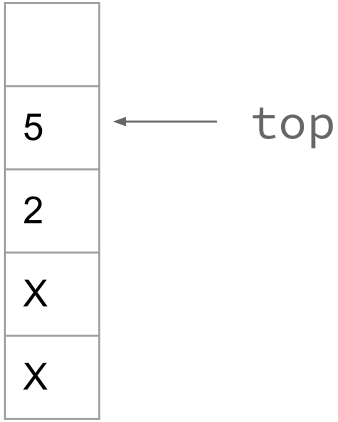
push(2)

Implementation design



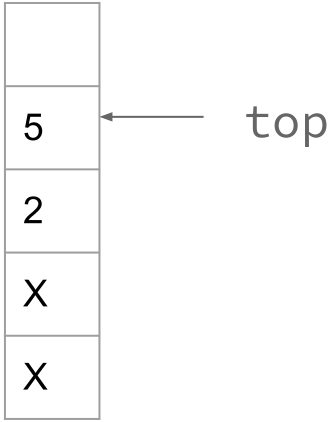

```
push(2)
push(5)
```

Implementation design



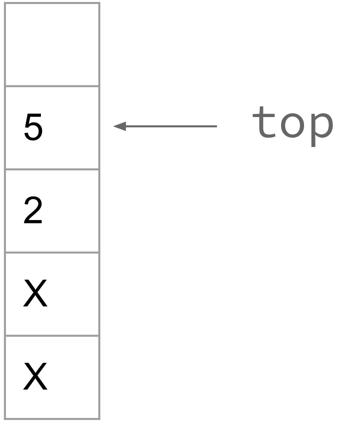

```
push(2)
push(5)
```

Implementation design



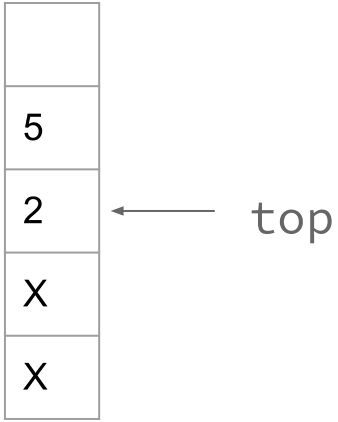

```
push(2)
push(5)
pop()
```

Implementation design



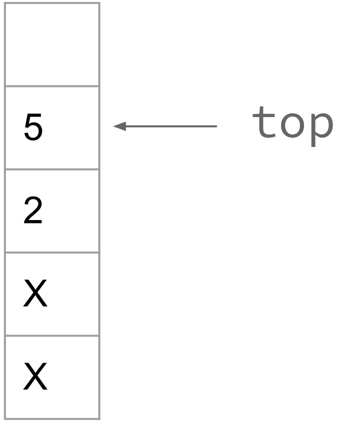

```
push(2)
push(5)
pop()
```

Implementation design



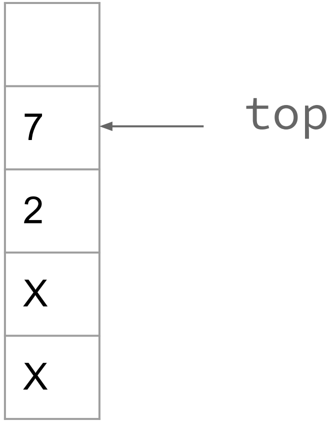

```
push(2)
push(5)
pop() //5 is returned
```

Implementation design



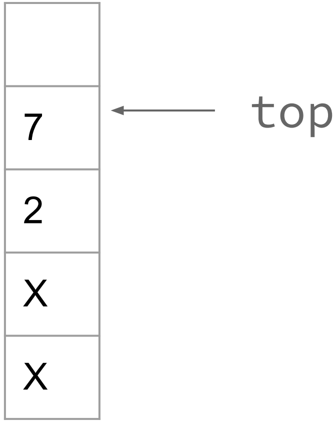

```
push(2)
push(5)
pop() //5 is returned
push(7)
```

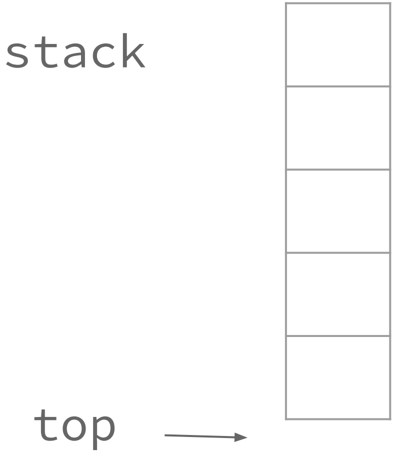
Implementation design




```
push(2)
push(5)
pop() //5 is returned
push(7)
```

Implementation design



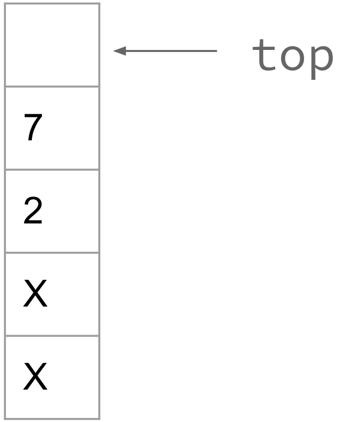


top = -1

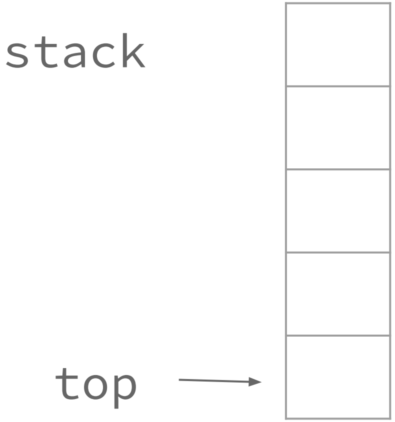
```
push(2)
push(5)
pop() //5 is returned
push(7)
```

```
push (int elem) {
    top++;
    stack[top]= elem;
}
```

Implementation design




```
push(2)
push(5)
pop() //5 is returned
push(7)
```



top = 0

```
push (int elem) {
    top++;
    stack[top]= elem;
}
```

```
int pop() {
  e = stack[top];
  top --;
  return e;
}
```

Complexity

Operation	Complexity
Push	O(1)
Pop	O(1)

Advantage and Limitation

- **Advantages of Array-based Implementation Fast:**

all operations are completed in one step. No loops are needed: O(1)

- **Limitations of Array-based Implementation:**

You have to know the upper bound of growth and allocate memory accordingly. If the array is full and there is another *push* operation then you encounter an exception (error).