Complexity of queues

Enqueue

Array or linked list is O(1). It's constant because we have a tail pointer, just add and update. Until queue is full, then double array, or item not added.

Dequeue

O(1) for removing from linked list or circular array. O(n) if array shifting needed for non-circular.

Stacks

With queues, we want to maintain order as first-in, first-out.

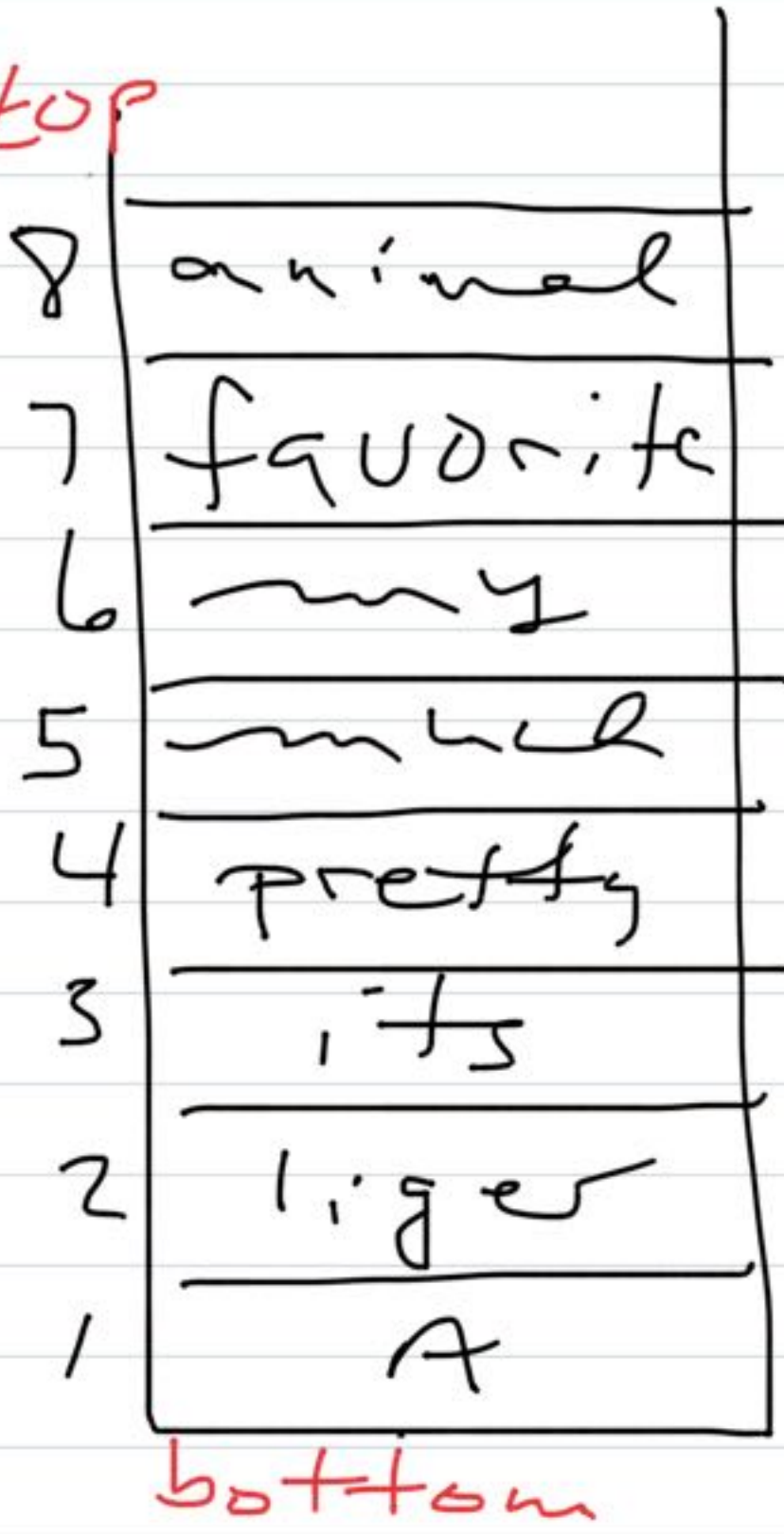
There is another data structure we use when we want to process data as last-in, first-out.

It's called a stack.

Stack is last-in, first-out, LIFO

Put words on stack

Add words from bottom to top, much like a stack of cafeteria plates



Remove from top only

animal favorite my much pretty its liger A.

When we add to stack, we say "push" onto the stack.

When we remove from the stack, we "pop" off the stack.

Always push and pop to top of stack.

We'll look at array and linked list implementation, but stack (and queue) doesn't have to be either.

Example: Where stacks are used.

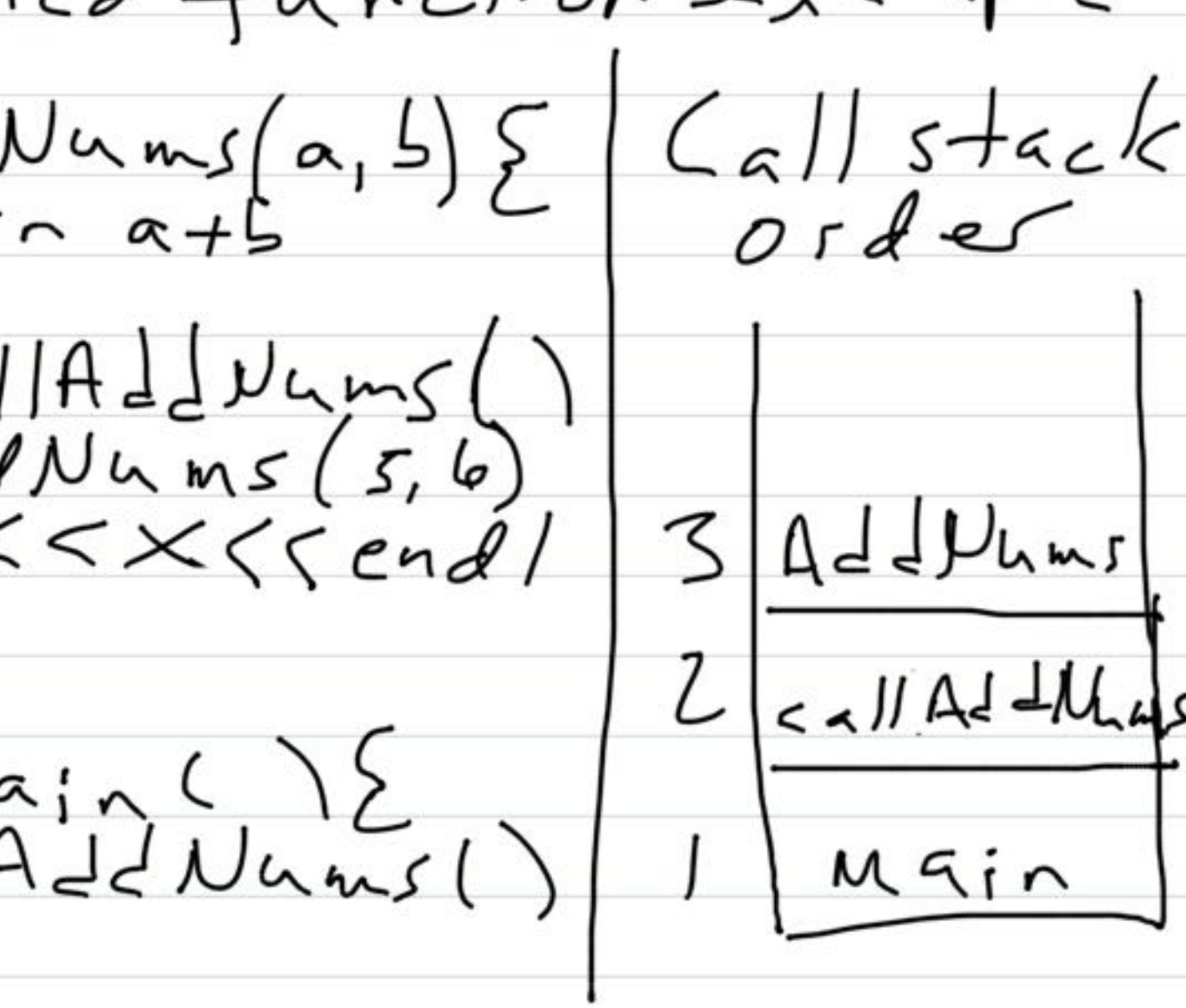
Computer Program execution

Commands pushed on to the stack, then popped off when executed.

Simplified function example

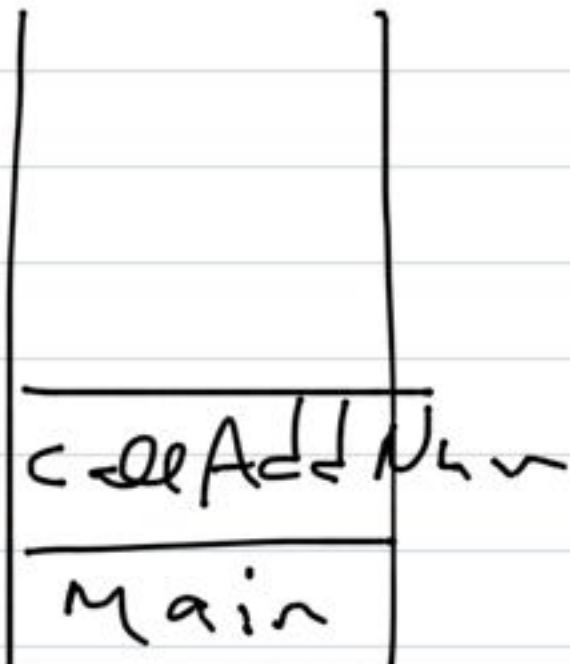
```
int addNums(a,b){
  return a+b
}
Void callAddNums()
  X=addNums(5,6)
  cout<<X<<endl
}

int main(){
  callAddNums()
}
```

Call stack order



Main added to stack first, then callAddNums, then AddNums.

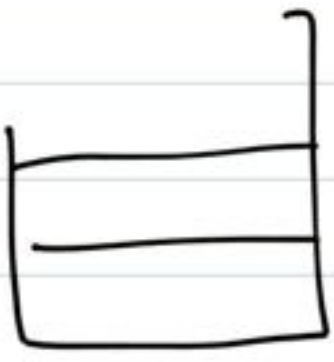
After AddNums done:



AddNums popped off stack. We return to caelAddNums

After callAddNums done, only main left on the stack. We return to main

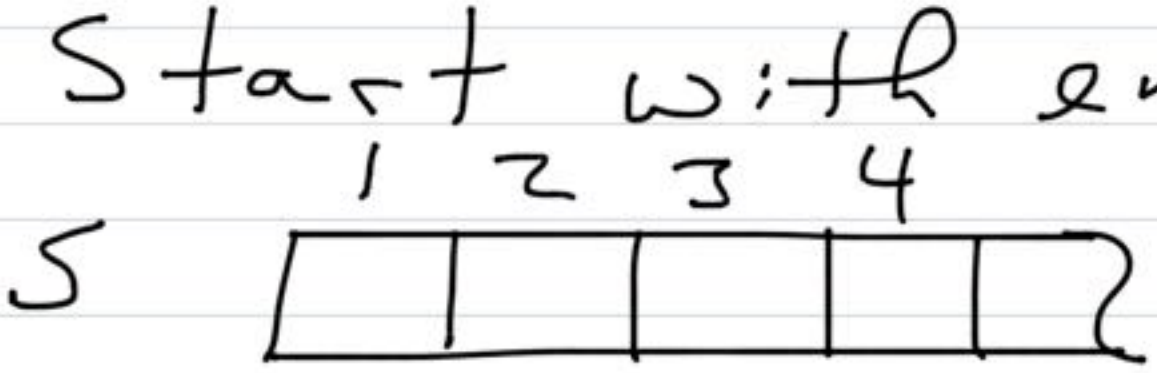
When main done, call stack is empty



Just like a queue, a stack can be implemented with an array or linked list, or any number of other ways.

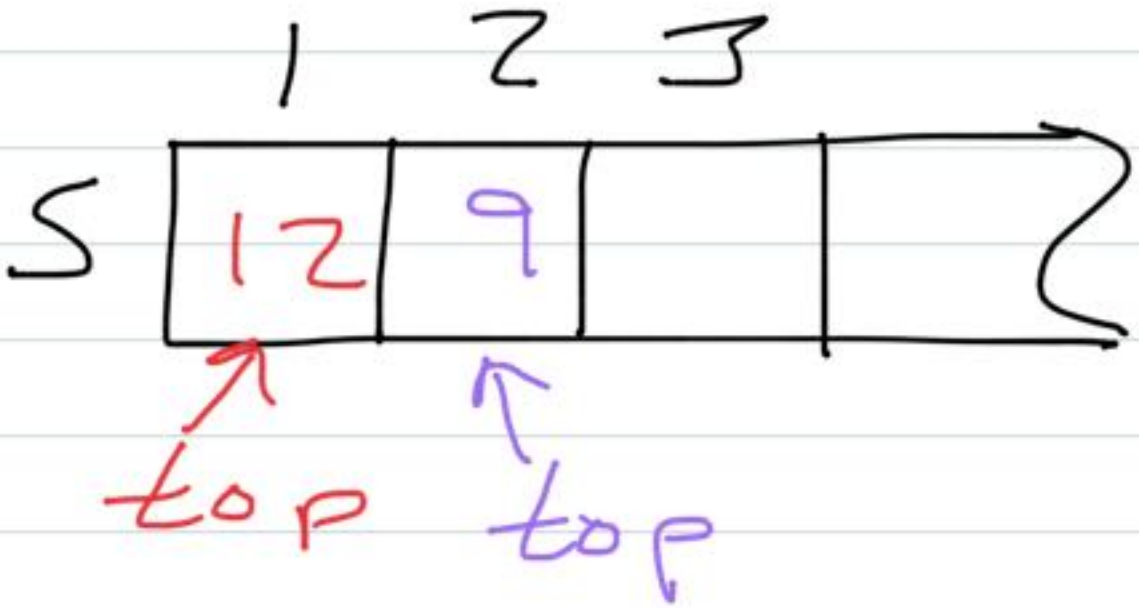
Array implementation

Start with empty stack



Pseudocode:

```
top = 0

Push(S, v)          // S is stack, v is value
    S.top = S.top + 1
    S[S.top] = v
```



top = 0

Push(12)

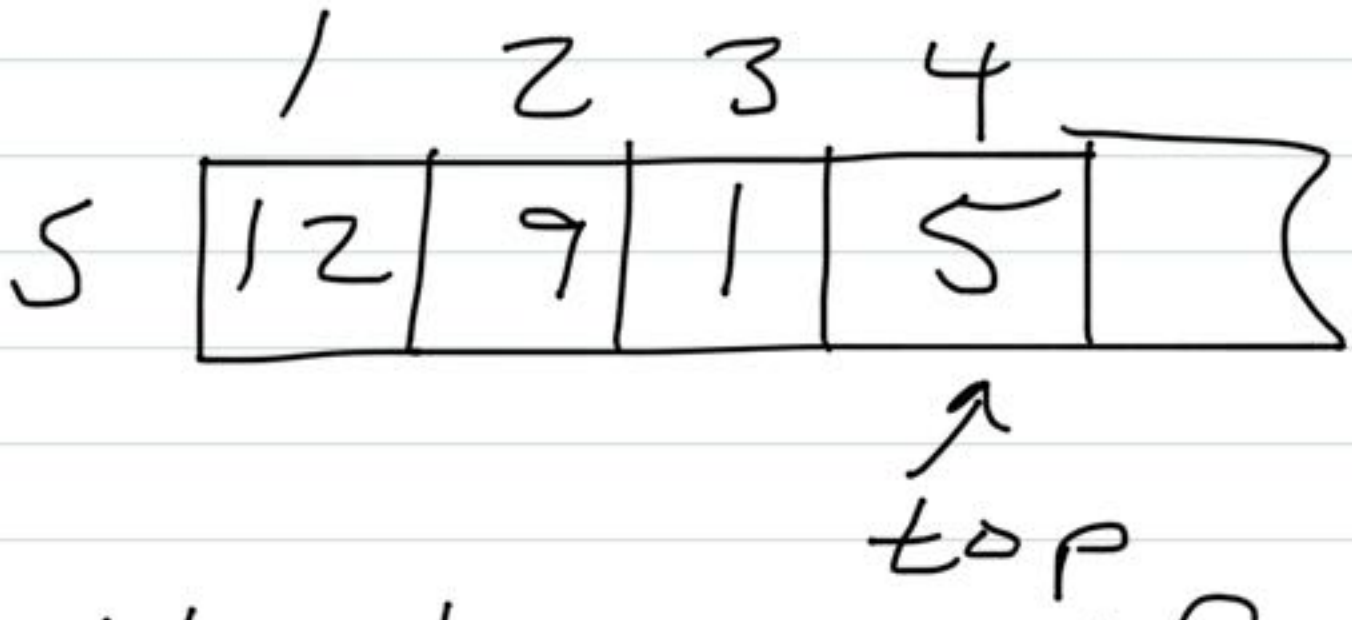
top = 1

s[1] = 12

Push(9)

top = 2

s[2] = 9



4 items on the stack

S[1 ... S.top] are contents of the stack.

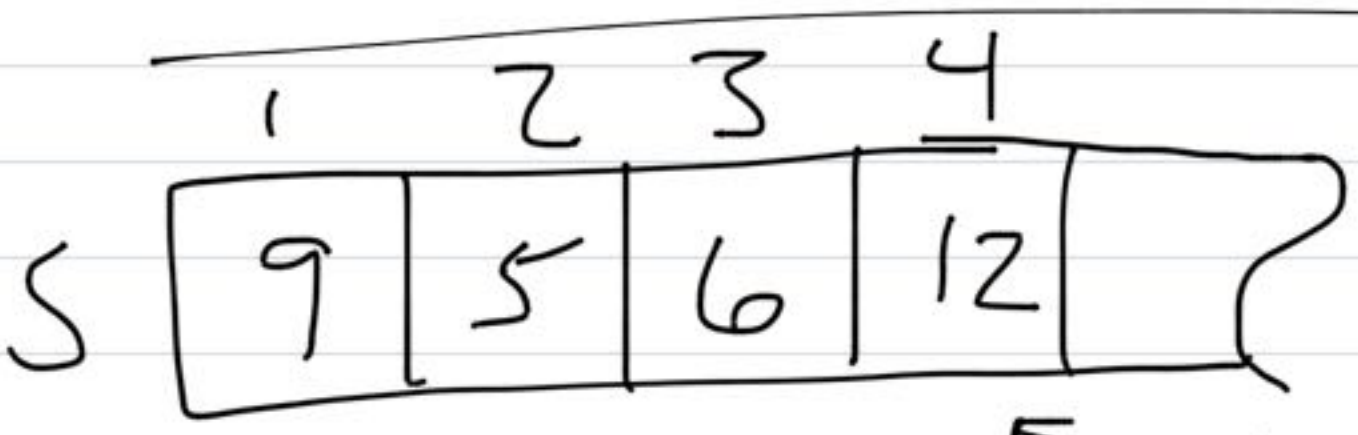
When S.top=0,
stack is empty

When S.top=n, stack is full (n is array size)

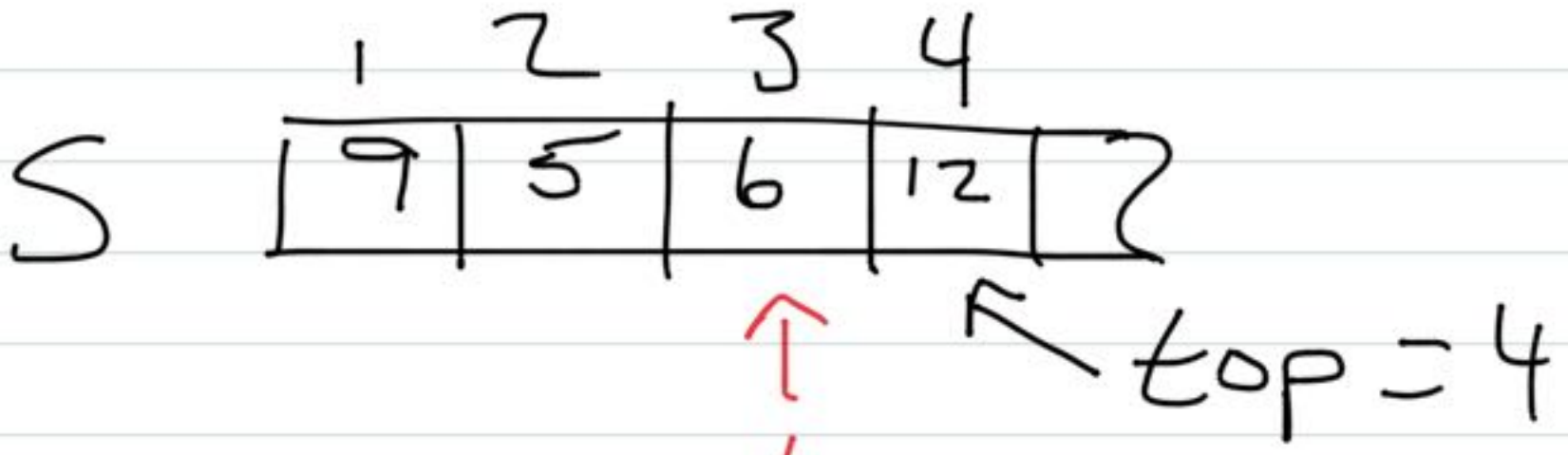
When S.top>n,
we say stack overflow.

Pseudocode doesn't address stack overflow

Remove from stack




```
Pop(S)
  if S.top == 0
      "underflow error"
  else
      top = top - 1
  return S[S.top + 1]
```



Pop(S)

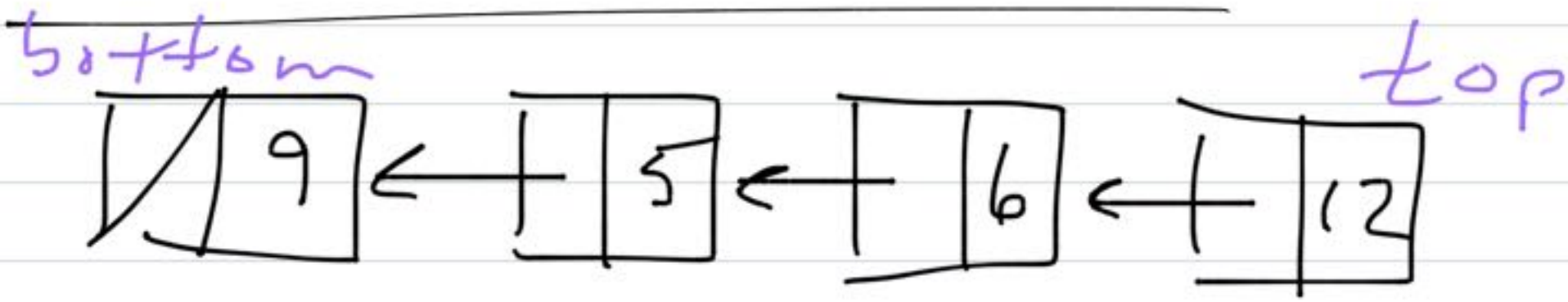
set top = 3

return S[4] //top+1

= 12

The algorithm in your book decrements the top first, then returns the value for where the top just was

Linked List implementation of a stack

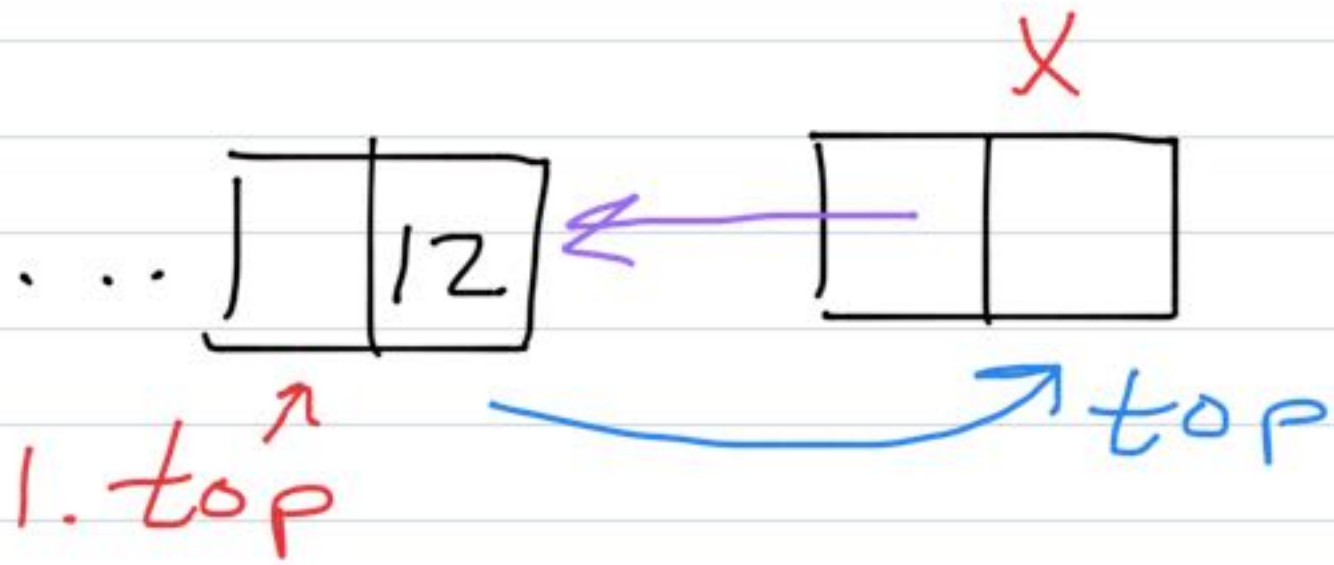


Each node contains link to the previous, Singly linked list

The node at the bottom has a previous pointer to NULL.

Push(L, x) // L is linked list, x is new node.

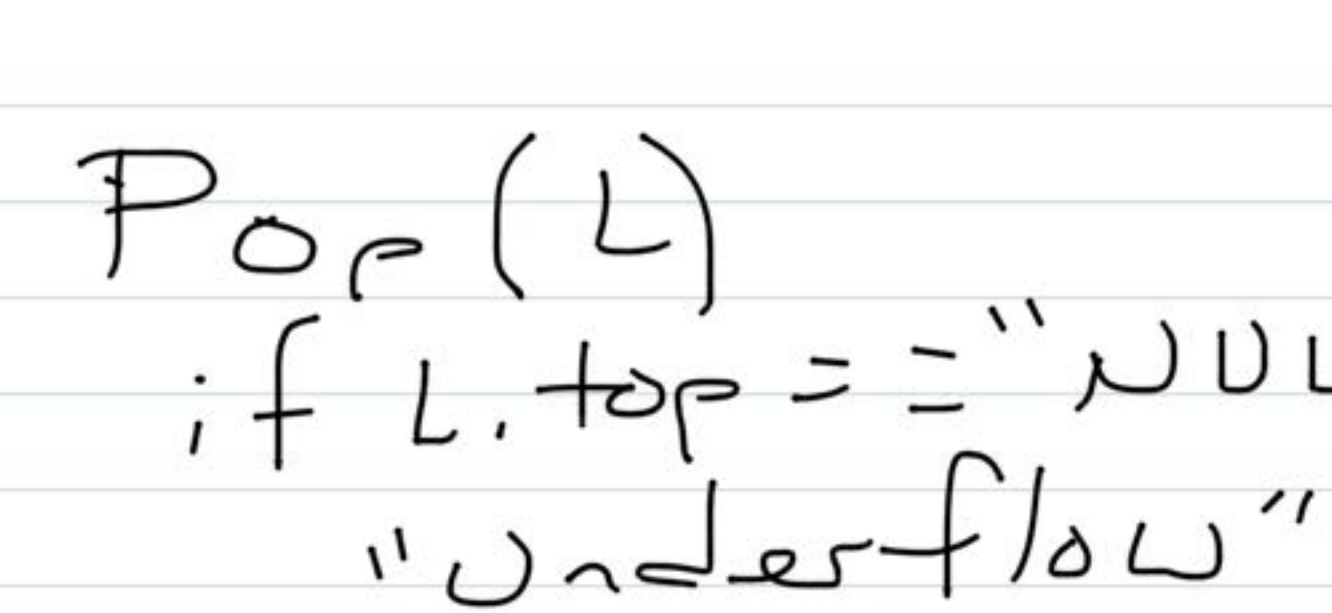
x.previous = L.top
L.top = x



2. Set x.previous = L.top
3. Move L.top to x

With a linked list, we're adding/removing a node, different than array where its a value.

```
Pop(L)
  if L.top == "NULL"
     "underflow"
  else
     x = L.top
     L.top = L.top.prev
  return x
```



x = L.top

Move L.top to L.top.prev

(Delete L.top)