Data Structures and Algorithms

Dr. Farshid Mehrdoust
Kiarash Dadpour

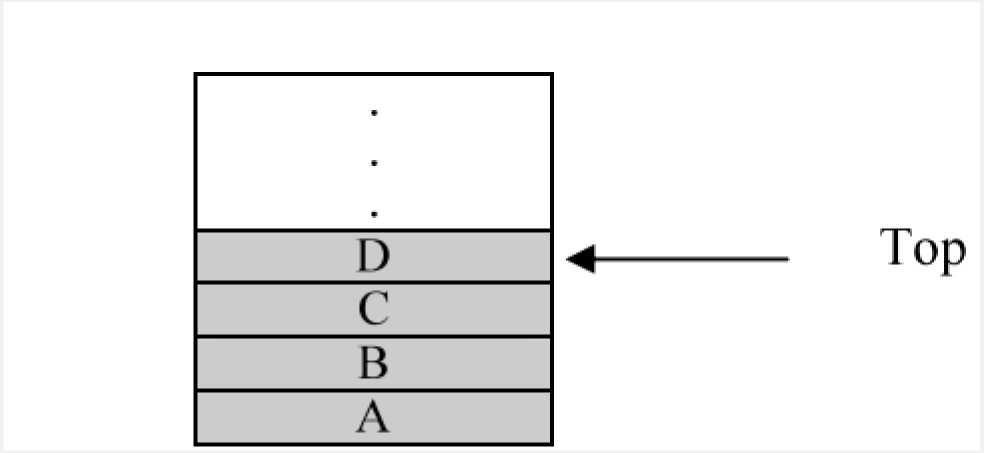
University of Guilan

03

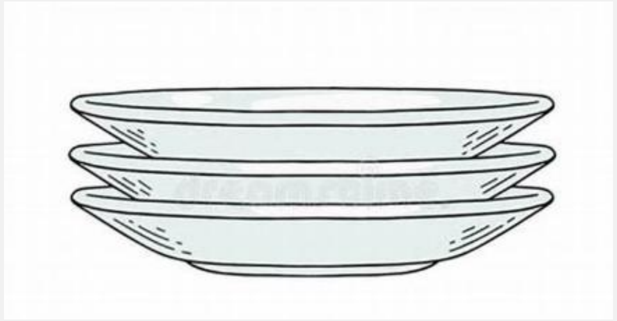
Stacks

Stack

A stack is a collection of objects that are inserted and removed according to the last-in, first-out (LIFO) principle.



Stack of Plates

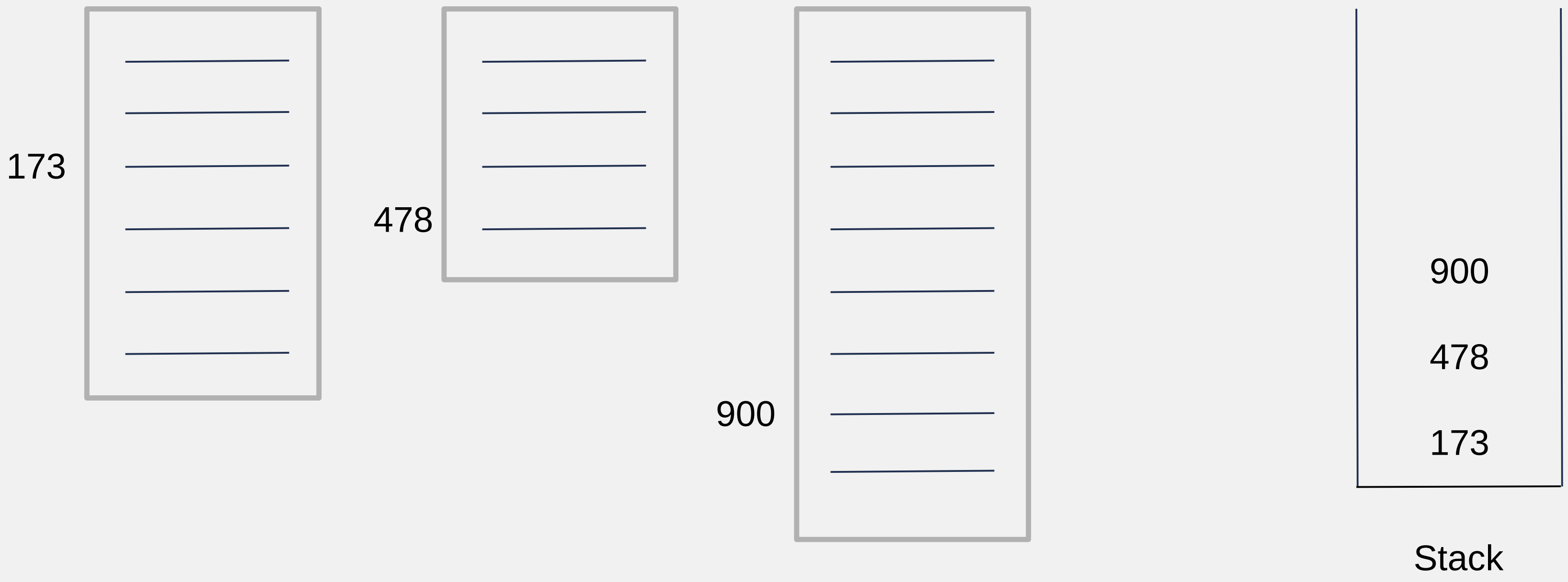


Examples

Internet Web browsers store the addresses of recently visited sites in a stack. Each time a user visits a new site, that site’s address is “pushed” onto the stack of addresses. The browser then allows the user to “pop” back to previously visited sites using the “back” button.

Text editors usually provide an “undo” mechanism that cancels recent editing operations and reverts to former states of a document. This undo operation can be accomplished by keeping text changes in a stack.

Stack in Recursive Functions



Stack Abstract Data Type

S.push(item)

Add element e to the top of stack S.

S.pop()

Remove and return the top element from the stack S;
an error occurs if the stack is empty.

S.top()

Return a reference to the top element of stack S, without removing it;
an error occurs if the stack is empty.

S.is_empty()

Return True if stack S does not contain any elements.

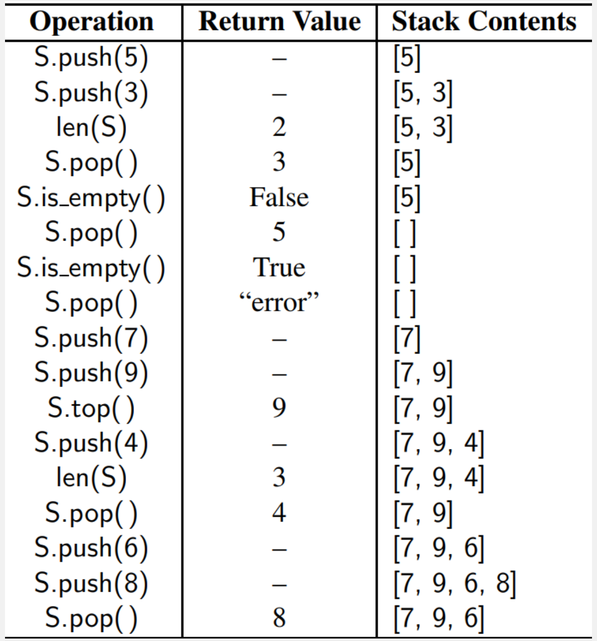
S.len()

Return the number of elements in stack S

Example

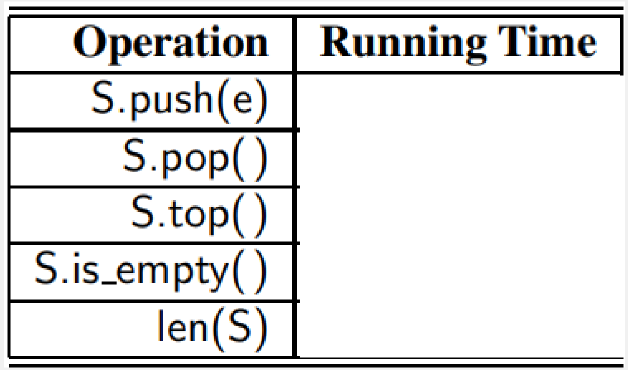
Operation	Return Value	Stack Contents
S.push(5)		
S.push(3)		
len(S)		
S.pop()		
S.is_empty()		
S.pop()		
S.is_empty()		
S.pop()		
S.push(7)		
S.push(9)		
S.top()		
S.push(4)		
len(S)		
S.pop()		
S.push(6)		
S.push(8)		
S.pop()		

Example



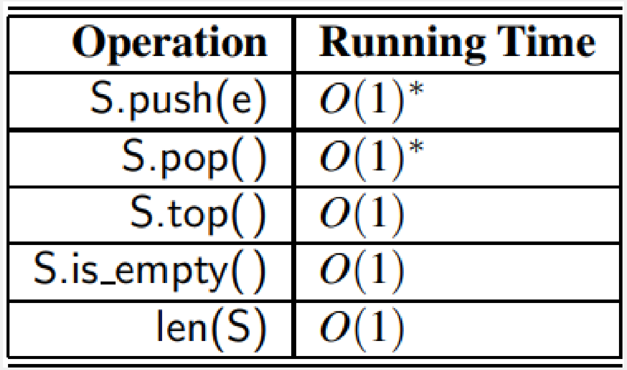
Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop()	3	[5]
S.is_empty()	False	[5]
S.pop()	5	[ ]
S.is_empty()	True	[ ]
S.pop()	“error”	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top()	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop()	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop()	8	[7, 9, 6]

Analyzing the Array-Based Stack Implementation



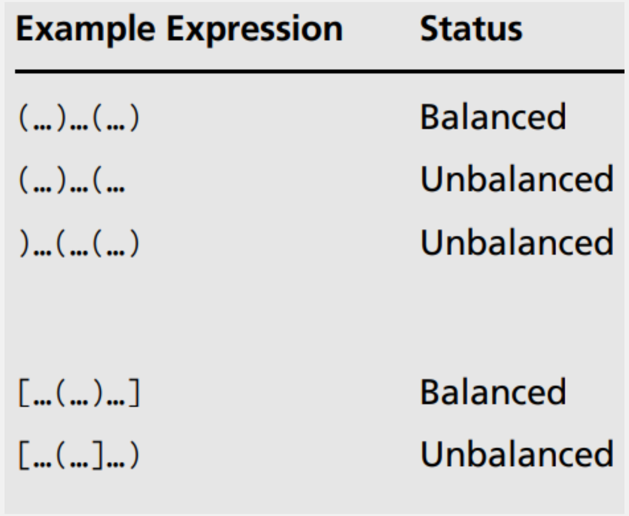
Operation	Running Time
S.push(e)	
S.pop( )	
S.top( )	
S.is_empty( )	
len(S)	

Analyzing the Array-Based Stack Implementation



Operation	Running Time
S.push(e)	$O(1)^*$
S.pop( )	$O(1)^*$
S.top( )	$O(1)$
S.is_empty( )	$O(1)$
len(S)	$O(1)$

Matching Parentheses



Example Expression	Status
(…)…(…)	Balanced
(…)…(…	Unbalanced
)…(…(…)	Unbalanced
[…(…)…]	Balanced
[…(…]…)	Unbalanced

Matching Parentheses – Examples

a) ((( )) (( ) (( ))))

b) [({ (({ (( ))) [ )] } {] }

c) { [( ) ( ]) [( )] }

d) { [( ) [( )] { ([ ]) } [ ] }

Stack-Generable Sequence

A sequence is called stack-generable if it can be produced by some sequence of push and pop operations.

Each number is first pushed onto the stack, and at any time the machine may pop from the stack to output a number.

Stack-Generable Sequence - Examples

Sequence:

1, 3, 4, 5, 7

a) 7, 5, 4, 1, 3
b) 1, 3, 7, 5, 4
c) 1, 4, 3, 5, 7

Stack-Generable Sequence - Examples

Sequence:

1, 2, 3, 4, 5, 6, 7

a) 4, 5, 3, 6, 1, 2, 7
b) 4, 3, 7, 6, 2, 5, 1
c) 2, 4, 5, 3, 7, 6, 1

Stack-Generable Sequence - Examples

Sequence:

1, 2, 3, 4, 5, 6, 7, 8, 9

a) 4, 5, 3, 2, 1, 9, 7, 6, 8
b) 3, 6, 5, 4, 1, 2, 9, 8, 7
c) 5, 4, 3, 2, 1, 6, 9, 8, 7

Convert Infix to Postfix

Infix:	a + b * c
Postfix:	
Infix:	a * b + d / e * f + g – m
Postfix:	
Infix:	a + b + (c – d) * ((e / f) – g) + k
Postfix:	
Infix:	(a + b) * (c – d) – e * f
Postfix:	

Convert Infix to Postfix

Infix:	a + b * c
Postfix:	a b c * +
Infix:	a * b + d / e * f + g – m
Postfix:	a b * d e / f * + g + m –
Infix:	a + b + (c – d) * ((e / f) – g) + k
Postfix:	a b + c d – e f / g - * + k +
Infix:	(a + b) * (c – d) – e * f
Postfix:	a b + c d - * e f * -

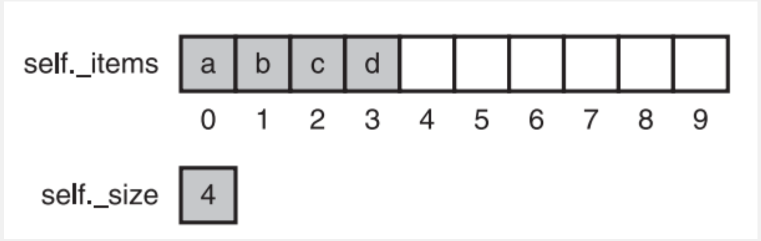
Convert Postfix to Infix

Postfix:	a b c * +
Infix:	
Postfix:	a b + d k * e / f / -
Infix:	
Postfix:	a b + c d - * e f * -
Infix:	

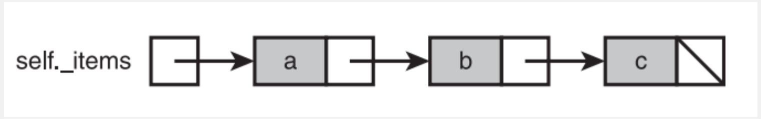
Convert Postfix to Infix

Postfix:	a b c * +
Infix:	a + b * c
Postfix:	a b + d k * e / f / -
Infix:	(a + b) – (((d * k) / e) / f)
Postfix:	a b + c d - * e f * -
Infix:	((a + b) * (c – d)) – (e * f)

Array-Based Stack Implementation



Linked List Based Stack Implementation



References

1) Fundamentals of Python: Data Structures / Kenneth A. Lambert

2) Introduction to algorithms / Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.

3) Data Structures and Algorithms in Python / Michael T. Goodrich, Robert Tamassia, Michael H. Goldwasser