Computer Networking Systems Department

Data structure

Daniah Abdul Qahar Shakir

2024-2025

Lecture 5

First course

A Stack: is a linear data structure that follows the LIFO (Last-In-First-Out) principle. Stack has one end. It contains only one pointer top pointer pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack.

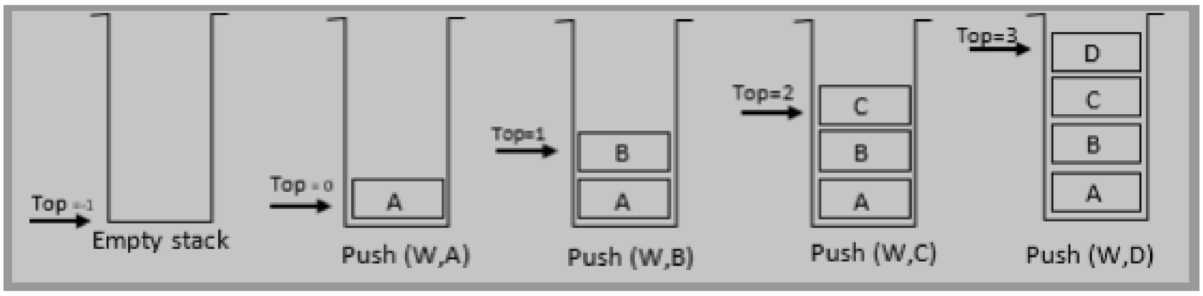
In computer science, a **stack** is an abstract data type that serves as a collection of elements, with two main operations:

- **Push**, which adds an element to the collection,
- **Pop**, which removes the most recently added element that was not yet removed.

Additionally, a peek operation can, without modifying the stack, return the value of the last element added.

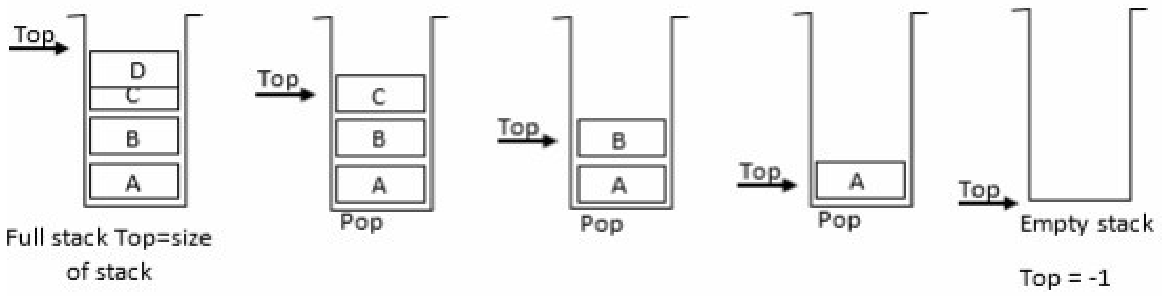
:Push operation

Let stack W[4], push A,B,C,D to the stack



:Pop operation

Let stack W[4], retrieve information from the stack



TYPES OF STACK

-Array (static: the size of stack is given initially

-Linked list (dynamic : never become full)

Push Algorithm

```
If(stack >=size-1)
  Then over flow
Else { top=top+1
       Stack[top]=new element}
```

Pop algorithm

```
If (top=-1)
Then the stack underflow
Else{  return stack [top]
          top=top-1
```

Notes: Top indicate to the top of stack which initialized (top =-1)

:**Application of Stack in real life**

- CD/DVD stand.
- Stack of books in a book shop.
- Call center systems.
- Undo and Redo mechanism in text editors.
- The history of a web browser is stored in the form of a stack.
- Call logs, E-mails, and Google photos in any gallery are also stored in form of a stack.
- YouTube downloads and Notifications are also shown in LIFO format(the latest appears first).
- Allocation of memory by an operating system while executing a process.

:Advantages of Stack

- **Easy implementation:** Stack data structure is easy to implement using arrays or linked lists, and its operations are simple to understand and implement.
- **Efficient memory utilization**: Stack uses a contiguous block of memory, making it more efficient in memory utilization as compared to other data structures.
- **Fast access time:** Stack data structure provides fast access time for adding and removing elements as the elements are added and removed from the top of the stack.
- **Helps in function calls:** Stack data structure is used to store function calls and their states, which helps in the efficient implementation of recursive function calls.
- **Supports backtracking:** Stack data structure supports backtracking algorithms, which are used in problem-solving to explore all possible solutions by storing the previous states.
- **Used in Compiler Design:** Stack data structure is used in compiler design for parsing and syntax analysis of programming languages.
- **Enables undo/redo operations**: Stack data structure is used to enable undo and redo operations in various applications like text editors, graphic design tools, and software development environments.

:Disadvantages of Stack

- **Limited capacity:** If the stack becomes full, adding new elements may result in stack overflow, leading to the loss of data.
- **No random access:** it only allows for adding and removing elements from the top of the stack. To access an element in the middle of the stack, all the elements above it must be removed.
- **Not suitable for certain applications:** Stack data structure is not suitable for applications that require accessing elements in the middle of the stack, like searching or sorting algorithms.
- **Stack overflow and underflow**: Stack data structure can result in stack overflow if too many elements are pushed onto the stack, and it can result in stack underflow if too many elements are popped from the stack.
- **Recursive function calls limitations:** While stack data structure supports recursive function calls, too many recursive function calls can lead to stack overflow, resulting in the termination of the program.

ch	opstack	Postfix	commentary
	#		Push # to opstack read ch
A	# #		Add ch to postfix read ch
+	# #+		Push + to opstack read ch
(	#+ #+(		Push (to opstack read ch
B	+ (+ (	 B	Add ch to postfix read ch
/	+ (+ (/	B B	Push / to opstack read ch
C	+ (/ + (/	B B C	Add ch to postfix read ch
)	+ (/ +	B C A B C /	Pop and add to postfix until (is reached.
#	+	B C / B C / + #	Pop and add to postfix until the opstack is empty.

#A*B-C(1+4-2)^D/W

CH	OPSTACK	POSTFIX	COMMENT
A	#	A	ADD A
*	#*	A	PUSH *
B	#*	AB	ADD B
-	# #-	AB* AB*	POP * PUSH -
C	#-	AB*C	ADD C
(	#-(	AB*C	PUSH (
1	#-(	AB*C1	ADD 1
+	#-(+	AB*C1	PUSH +
4	#-(+	AB*C14	ADD 4
-	#-(#-(-	AB*C14+ AB*C14+	POP + PUSH -
2	#-(-	AB*C14+2	ADD 2
)	#-(- #-	AB*C14+2 AB*C14+2-	POP -
^	#-^	AB*C14+2-	PUSH ^
D	#-^	AB*C14+2-D	ADD D
/	#-^ #-/	AB*C14+2-D^	POP ^ PUSH /
W	#-/	AB*C14+2-D^W	ADD W
#		AB*C14+2- D^W/-	POP ALL STACK

-/OUTPUT STRING :AB*C14+2-D^W

References

- Introduction to Algorithms, 3rd Edition by *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein*

- Introduction to Algorithms, 3rd Edition by *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein*

- Elements of Programming Interviews in Java: The Insiders' Guide, by *Adnan Aziz, Tsung-Hsien Lee, Amit Prakash*

- https://github.com/careermonk/DataStructuresAndAlgorithmsMadeEasy