FullStack.Cafe - Kill Your Tech Interview

Q1: Define Stack ☆

Topics: Stacks Data Structures

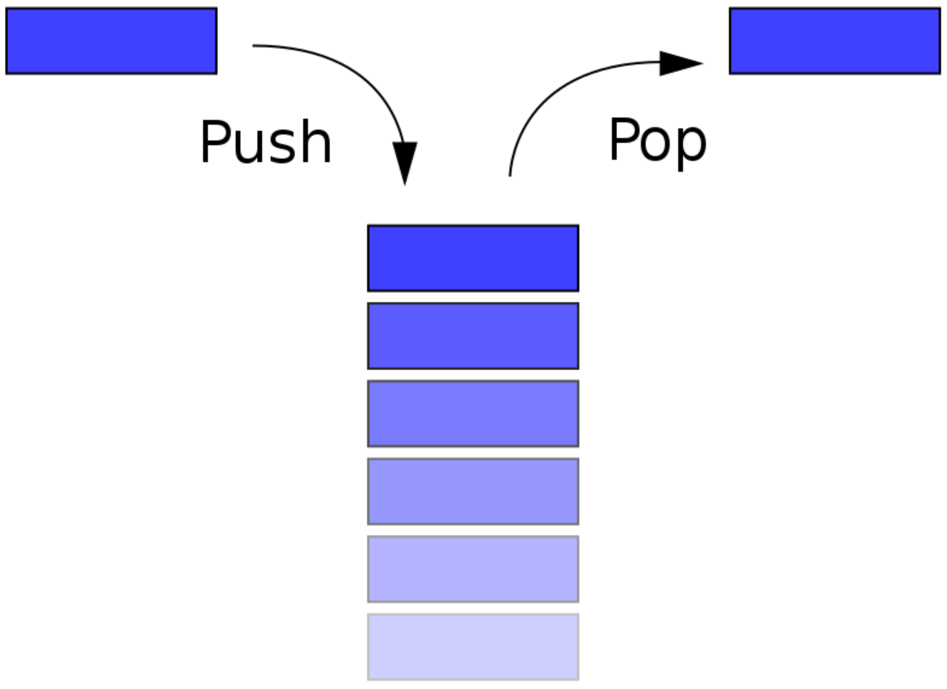
Answer:

A **Stack** is a container of objects that are inserted and removed according to the last-in first-out (**LIFO**) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack.

There are basically three operations that can be performed on stacks. They are:

1. inserting an item into a stack (**push**).
2. deleting an item from the stack (**pop**).
3. displaying the contents of the stack (**peek** or **top**).

A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.



Q2: Explain why Stack is a recursive data structure ☆

Topics: Stacks Data Structures

Answer:

A **stack** is a **recursive** data structure, so it's:

- a stack is either empty or

- it consists of a top and the rest which is a stack by itself;

Q3: Why and when should I use Stack or Queue data structures instead of Arrays/Lists? ☆☆

Topics: Queues Stacks Data Structures

Answer:

Because they help manage your data in more a *particular* way than arrays and lists. It means that when you're debugging a problem, you won't have to wonder if someone randomly inserted an element into the middle of your list, messing up some invariants.

Arrays and lists are random access. They are very flexible and also easily *corruptible*. If you want to manage your data as FIFO or LIFO it's best to use those, already implemented, collections.

More practically you should:

- Use a queue when you want to get things out in the order that you put them in (FIFO)
- Use a stack when you want to get things out in the reverse order than you put them in (LIFO)
- Use a list when you want to get anything out, regardless of when you put them in (and when you don't want them to automatically be removed).

Q4: How to implement Linked List Using Stack? ☆☆

Topics: Linked Lists Stacks

Answer:

You can simulate a linked list by using two stacks. One stack is the "list," and the other is used for temporary storage.

- To **add** an item at the head, simply push the item onto the stack.
- To **remove** from the head, pop from the stack.
- To **insert** into the middle somewhere, pop items from the "list" stack and push them onto the temporary stack until you get to your insertion point. Push the new item onto the "list" stack, then pop from the temporary stack and push back onto the "list" stack. Deletion of an arbitrary node is similar.

This isn't terribly efficient, by the way, but it would in fact work.

Q5: Why Are Stacks Useful? ☆☆

Topics: Stacks

Answer:

They’re very useful because they afford you constant time `O(1)` operations when *inserting* or *removing* from the front of a data structure. One common use of a stack is in compilers, where a stack can be used to make sure that the brackets and parentheses in a code file are all balanced, i.e., have an opening and closing counterpart. Stacks are also very useful in evaluating mathematical expressions.