Stack

A data structure that provides the following function in O(1) time:

- Push: put element at the top of the Stack
- Pop: return element from top of the Stack and removes that element from Stack
- Peek: return element from top without removing it.
- isEmpty: return true if the Stack is empty

Implementation

Based on a dynamic array.
Put element in pre defined sized array. Keep track of the top element by updating the variable that holds the index to the last element. Once the top variable reaches capacity, it creates a new array by doubling the size of the original array, and copy all previous element to the new one.

Based on a singly LinkedList
Every time an element is pushed to the Stack, a new Node is created and holds the element that is being pushed. The head pointer gets updated to point to the newly created Node.

Differences of the two implementation

Array implementation	LinkedList implementation
• Better locality. For a small enough size array, the entire array might be already loaded to the cache, making the access to the element much faster. • Worst case can be O(n), as array need to be resized if initial capacity is exceeded.	• Relink of the Node is always O(1) even for worst cases. However, every time an object is pushed, a new object is created. Allocation of new object is more expensive than regular operations. • Takes more memory. Each Node contains the data element and a pointer (an address, 8bytes for 64 bit) to the next Node.

Some applications

- Reverse a word
- “undo” function in text editors, where text changes are kept in a Stack (Backtracking)
- matching symbols: matching braces, used in calculators. (Infix, Prefix, PostFix)