Basic Operations

- **Push**
  - inserts one element onto the stack

- **Pop**
  - returns the element at the top of the stack (and removes it)

- **IsEmpty**
  - not necessary, but sometimes useful

**LIFO: Last In First Out**
Implementation

- Arrays
  - push and pop at the end of the array (easier and efficient)
  - can be fixed-length
  - can also use a dynamic array (grows over time)
    - additional cost for dynamic arrays
  
  ![Array Diagram](https://www.cs.usfca.edu/~galles/visualization/StackArray.html)

Considerations

- Underflow
  - error can be thrown when calling pop on an empty stack
- Overflow
  - error can be thrown when calling push on a full stack
    - especially in fixed-length implementations

Applications

- Undo in software applications
- Navigation buttons in browsers
- Stack in compilers/programming languages
- Parsing expressions
  - ...

Implementation

- Linked Lists
  - push and pop at front (could use the other end as well)

  ![Linked List Diagram](https://www.cs.usfca.edu/~galles/visualization/StackLL.html)
Example

- Fully parenthesized infix expressions
  - **infix arithmetic expressions**: operators are placed between two operands
  - **fully parenthesized infix expression**: infix arithmetic expression where every operator and its arguments are contained in parentheses
  - **operator precedence** and **associativity** don't matter

\[((5 + ((10 - 4) * (3+2))) + 25)\]

Dijkstra’s two stacks algorithm

<table>
<thead>
<tr>
<th>Element</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>operand (value)</td>
<td>push it onto the s1</td>
</tr>
<tr>
<td>operator</td>
<td>push it onto s2</td>
</tr>
<tr>
<td>left parenthesis</td>
<td>ignore</td>
</tr>
<tr>
<td>right parenthesis</td>
<td>pop operator from s2 and pop two values from s1, then apply operator to those values and push the result onto s1</td>
</tr>
</tbody>
</table>

(FIFO: First In First Out)

Queues
Basic Operations

- **Enqueue**
  - inserts one element onto the queue
- **Dequeue**
  - returns the next element from the queue (and removes it)
- **IsEmpty**
  - not necessary, but sometimes useful

Implementation

- **Arrays**
  - enqueue and dequeue at different ends of the array
  - can be fixed-length
  - can also use a dynamic array (grows over time)
    - additional cost for dynamic arrays

- **Linked Lists**
  - enqueue and dequeue at different ends

head \[\rightarrow\] 7 \[\rightarrow\] 3 \[\rightarrow\] 5

tail

https://www.cs.usfca.edu/~galles/visualization/QueueArray.html

https://www.cs.usfca.edu/~galles/visualization/QueueLL.html
### Considerations

- **Underflow**
  - error can be thrown when calling `dequeue` on an empty queue

- **Overflow**
  - error can be thrown when calling `enqueue` on a full queue (especially in fixed-length implementations)

### Applications

- Media Playlists (Youtube, Spotify, Music, etc.)
- Process management in Operating Systems
- Simulations
- Used in other algorithms
- …