CS350: Data Structures

Stacks

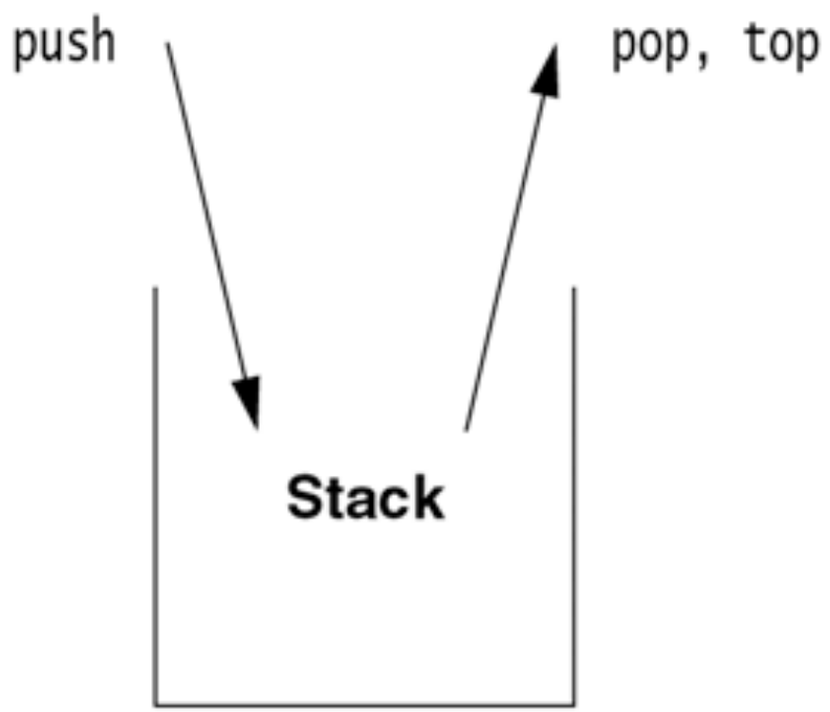
James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania

Stacks

- **Stacks are a very common data structure that can be used for a variety of data storage purposes**
- **Similar to a stack of papers sitting on a table ...**
  - Items can be placed on the stack of papers
  - You can take a look at whats on the top of your stack of papers
  - Items can be removed from the top of your stack of papers
  - If you want a paper in the middle of the stack, you must remove the papers on top to get to the paper in the middle
- **May also be referred to as a LIFO (Last-In-First-Out)**

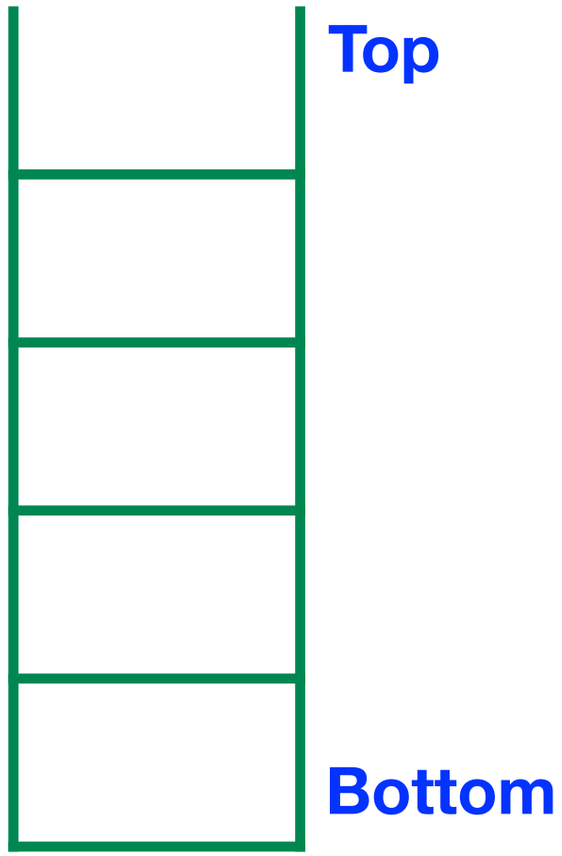
Stack Operations

- **Stacks have three main operations**
  - **`push`** - puts an element on the top of the stack
  - **`pop`** - removes a single element from the top of the stack
  - **`top`** or **`peek`** - returns the value of the element on the top of the stack (without removing it)



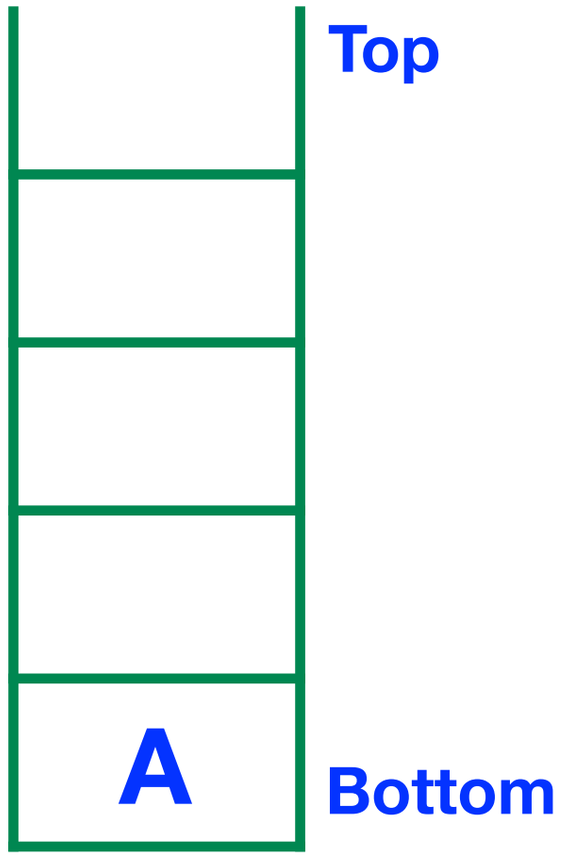
Stack Operations

Start with Empty Stack



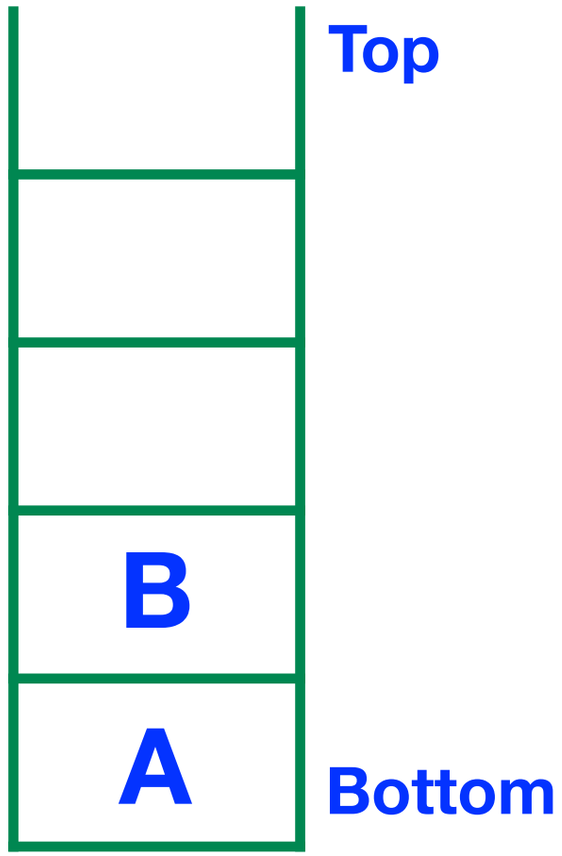
Stack Operations

Push Value: A



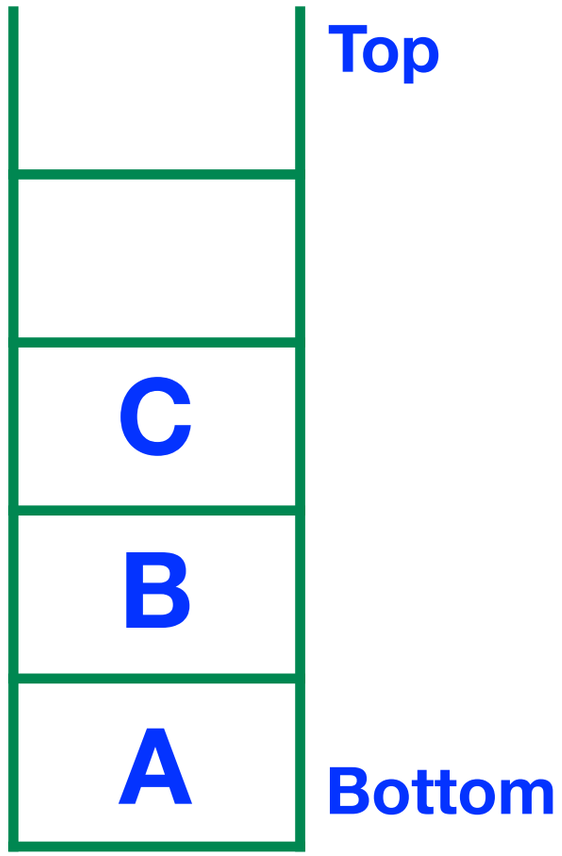
Stack Operations

Push Value: B

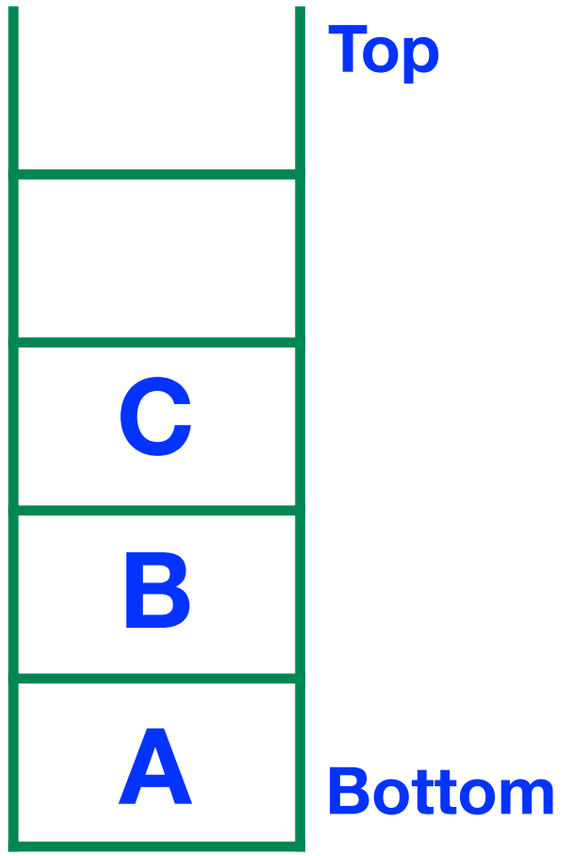


Stack Operations

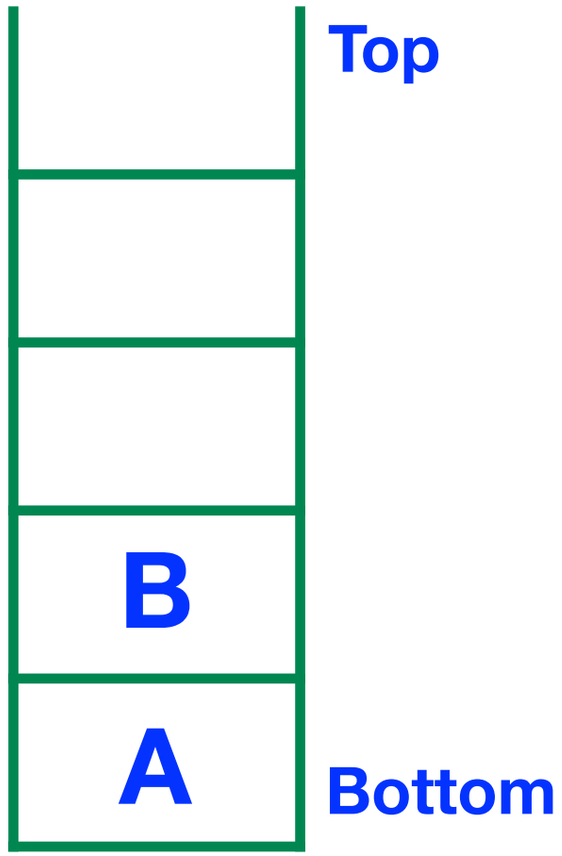
Push Value: C



Pop the Top of the Stack:



Pop the Top of the Stack:



A Stack Interface in Java

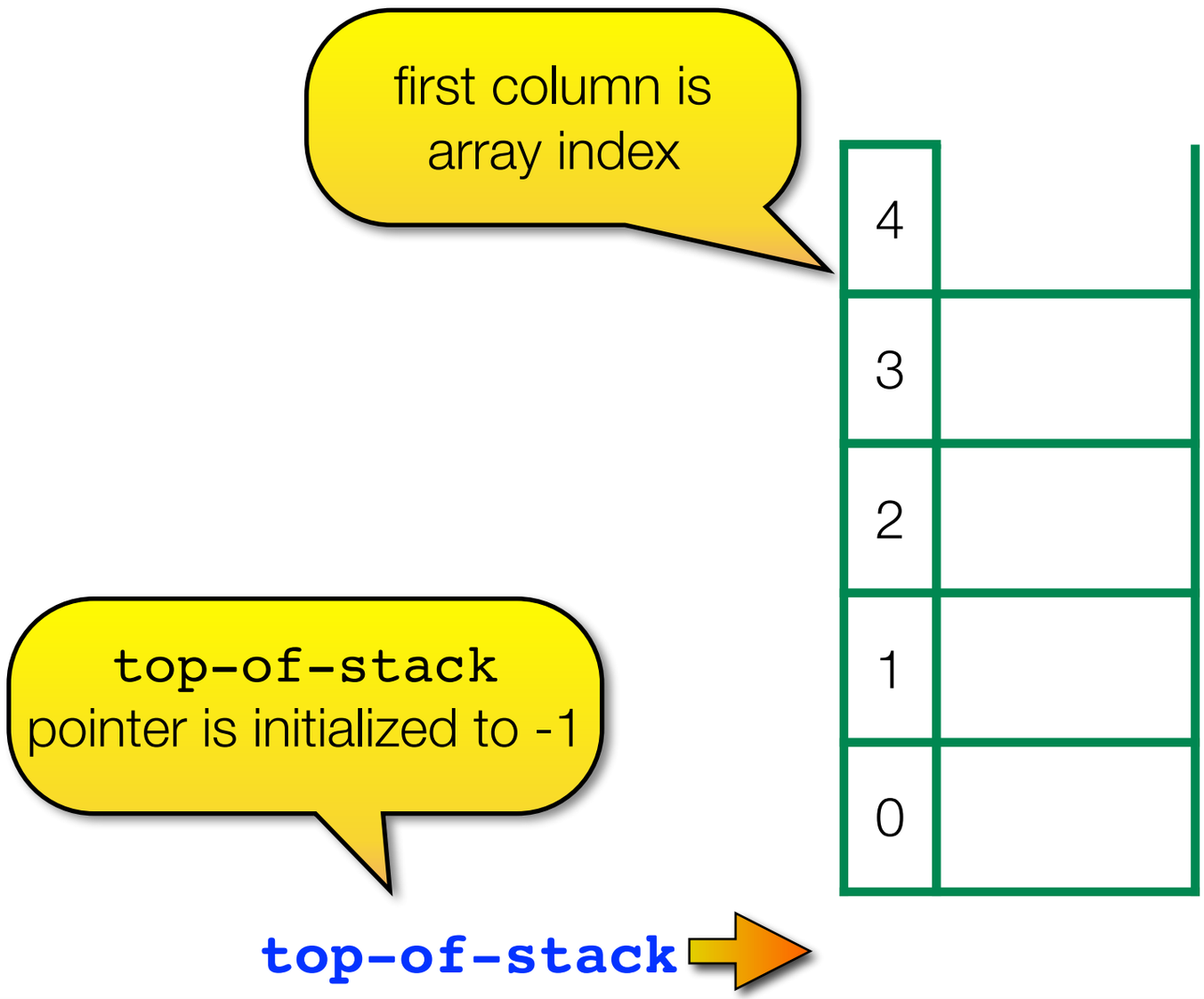
```
public interface Stack<AnyType> {

    public void    push( AnyType x );

    public AnyType pop( );

    public AnyType peek( );

    public boolean isEmpty( );

    public void    makeEmpty( );
}
```

Stack Implementations

- **Stacks can be implemented in multiple ways**
- **Two popular methods for implementing stacks include**
  - (1) Arrays
  - (2) Linked Lists
- **Both of these implementation approaches allow for constant time operations - *O*(1)**

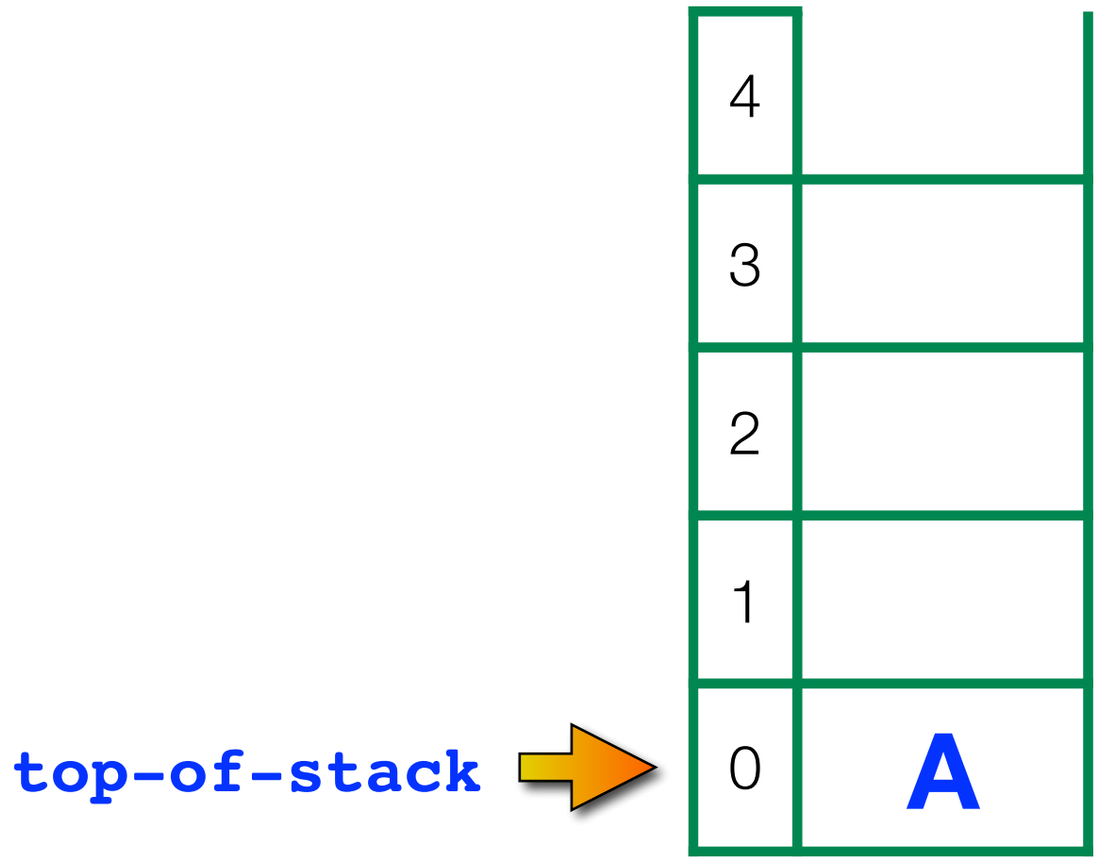
Stack Implementation Using an Array

Start with Empty Stack (i.e. an array)



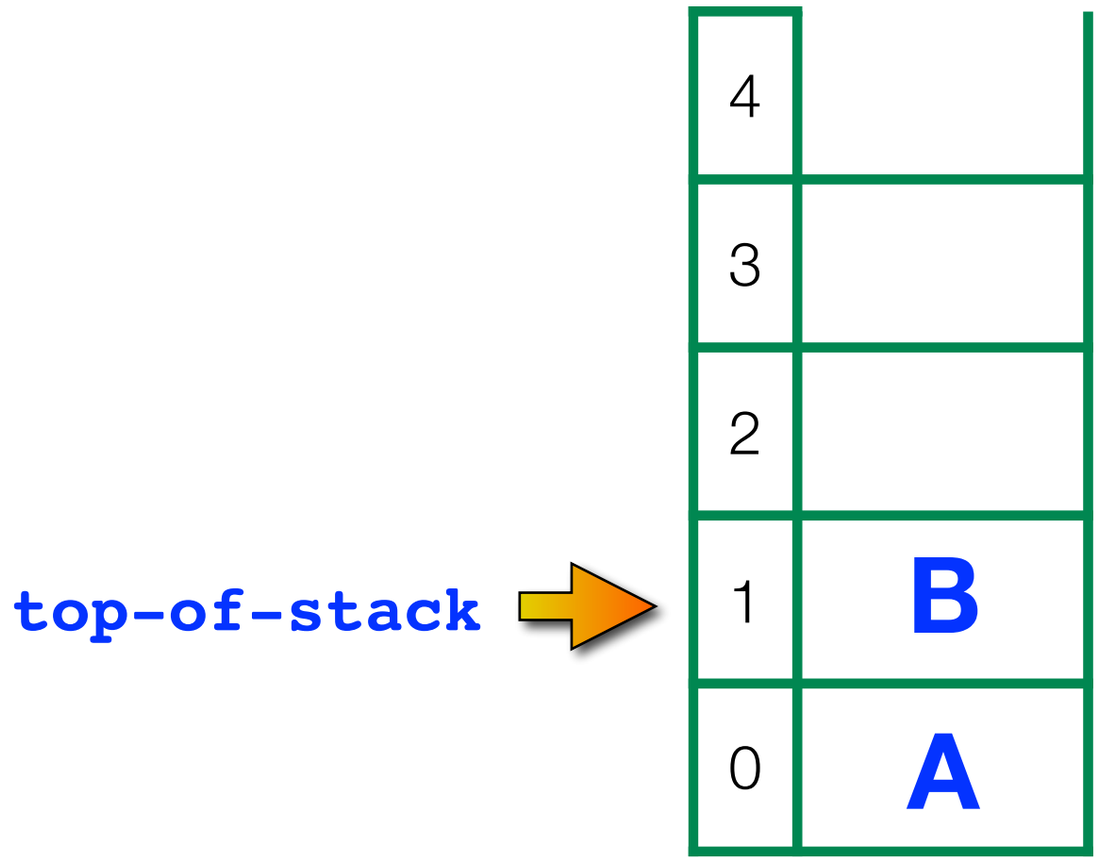
Stack Implementation Using an Array

Insert Value: A



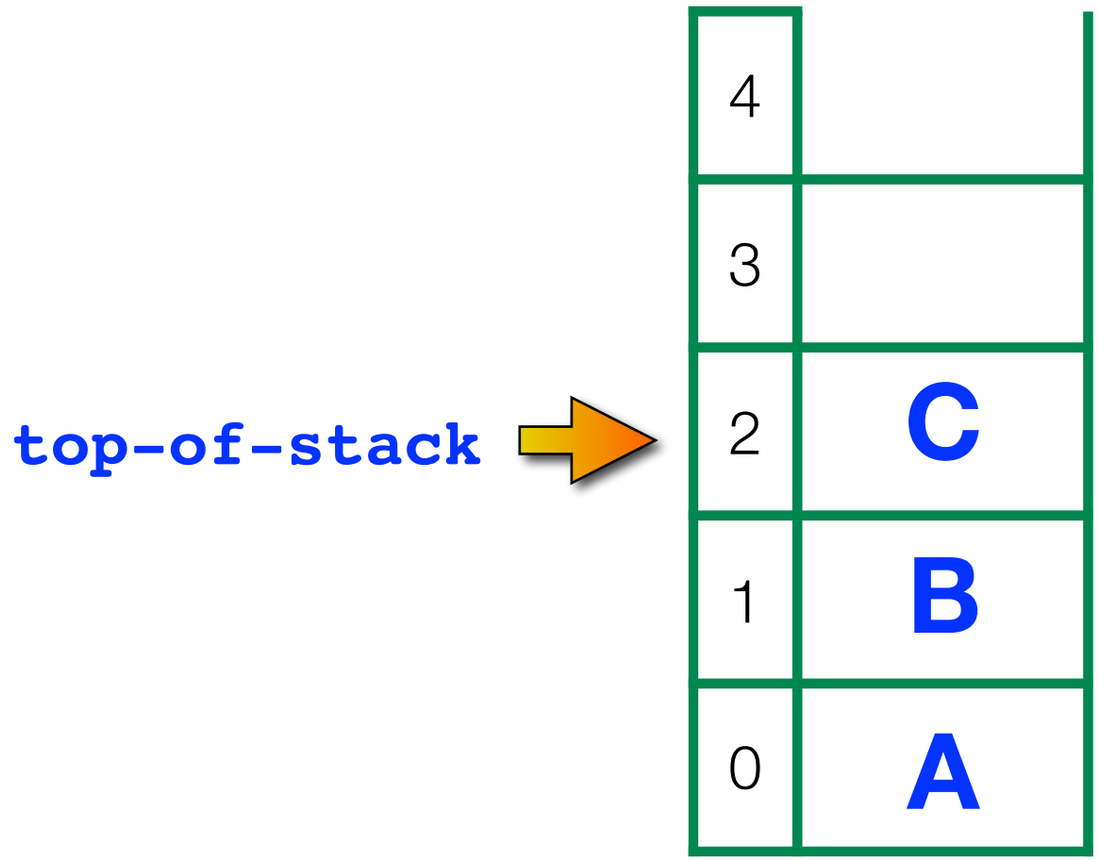
Stack Implementation Using an Array

Insert Value: B



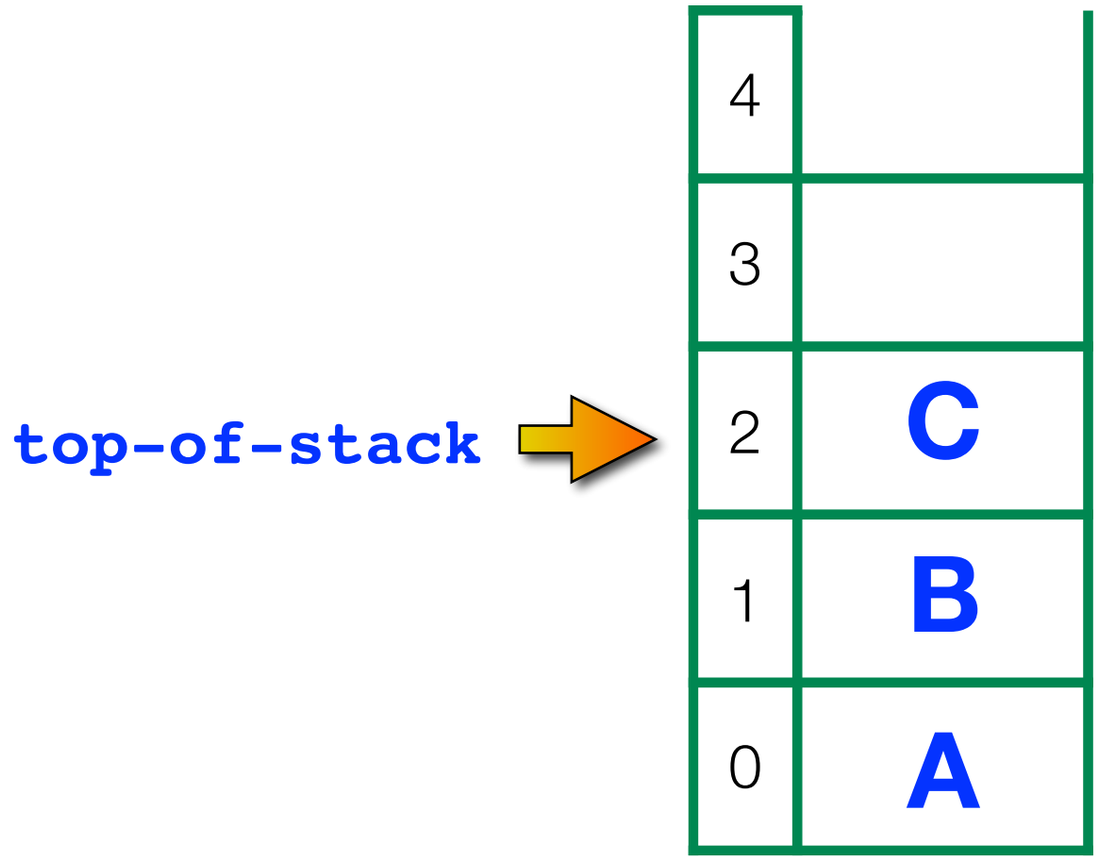
Stack Implementation Using an Array

Insert Value: C



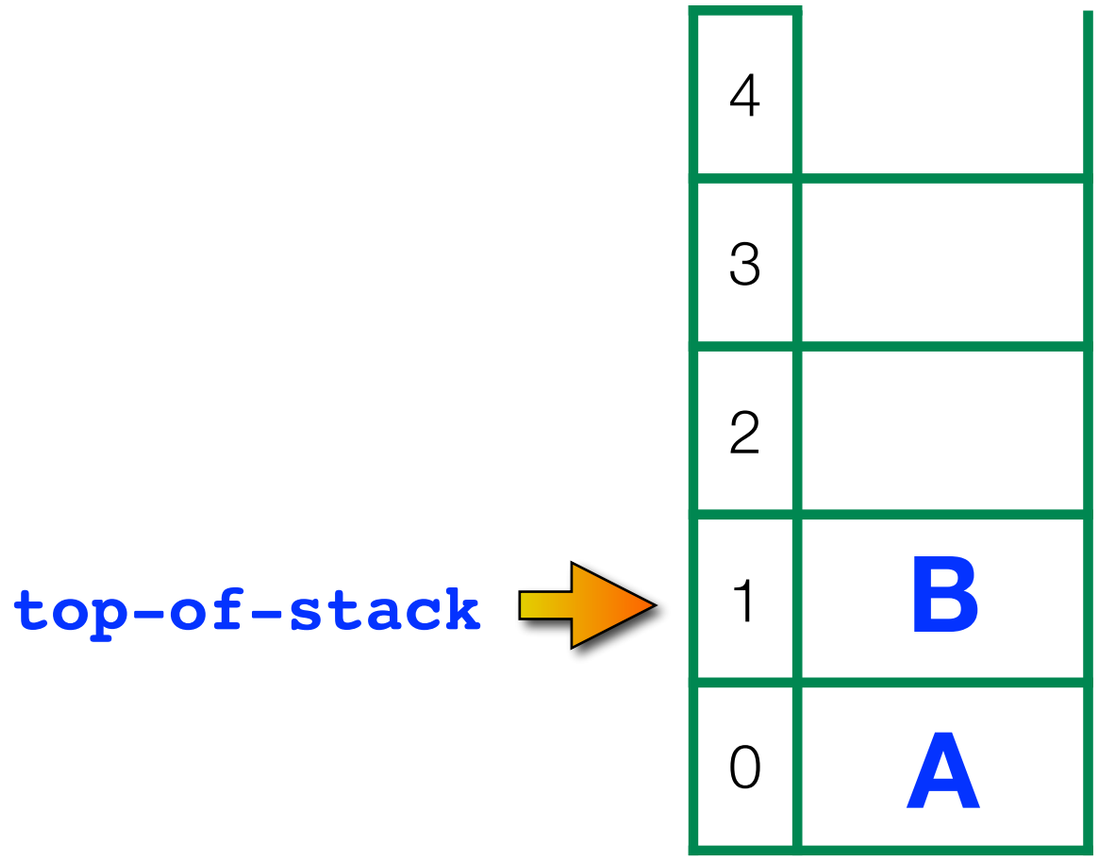
Stack Implementation Using an Array

Pop the Top of the Stack:



Stack Implementation Using an Array

Pop the Top of the Stack:



Stack Implementation Using an Array

- **Considerations when using an array implementation**
  - **`push`**, **`pop`**, and **`peek`** operations all run in constant time ... in most cases
  - What happens when your array is full and you want to **`push`** another element?
    - Array must be increased in size which takes time an more memory
    - Time to copy and create new array is *O*(N)
    - Time to copy array is amortized over the lifetime of the array
    - May not be suitable for all types of systems (e.g. RTOS)

Stack Implementation Using an Array (Java)

```
/**
 * Stack constructor
 */
public ArrayStack( )
{
    theArray = (AnyType [ ]) new Object[ DEFAULT_CAPACITY ];
    topOfStack = -1;
}
```

Stack Implementation Using an Array (Java)

```
/**
 * Test if the stack is logically empty.
 * @return true if empty, false otherwise.
 */
public boolean isEmpty( )
{
    return topOfStack == -1;
}
```

Stack Implementation Using an Array (Java)

```
/**
 * Make the stack logically empty.
 */
public void makeEmpty( )
{
    topOfStack = -1;
}
```

Stack Implementation Using an Array (Java)

```
/**
 * Insert a new item into the stack.
 * @param x the item to insert.
 */
public void push( AnyType x )
{
    if( topOfStack + 1 == theArray.length )
        doubleArray( );
    theArray[ ++topOfStack ] = x;
}
```

Stack Implementation Using an Array (Java)

```
/**
 * Remove the most recently inserted item from the stack.
 * @throws UnderflowException if the stack is empty.
 */
public void pop( )
{
    if( isEmpty( ) )
        throw new UnderflowException( "ArrayStack pop" );
    topOfStack--;
}
```

Stack Implementation Using an Array (Java)

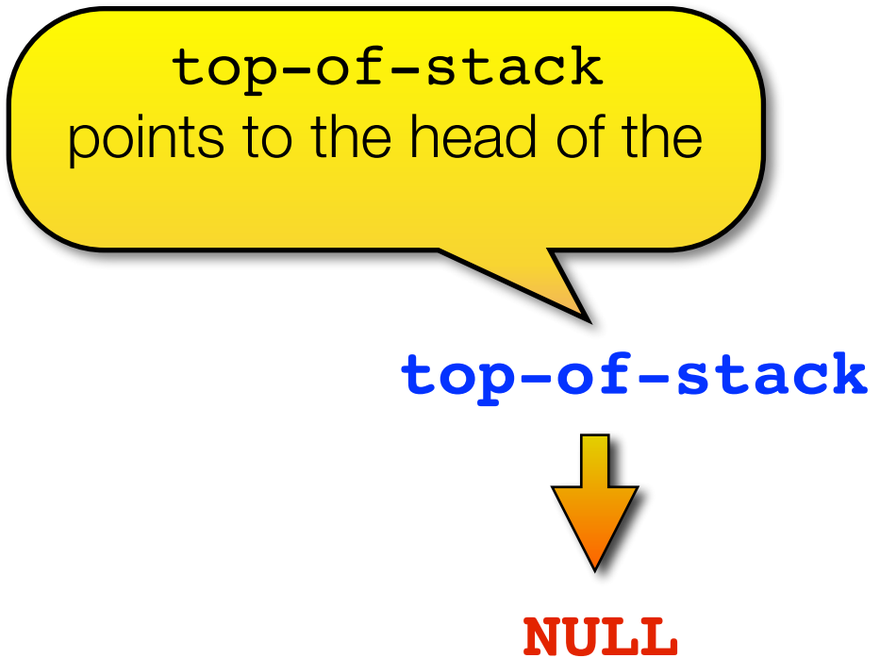
```
/**
 * Get the most recently inserted item in the stack.
 * Does not alter the stack.
 * @return the most recently inserted item in the stack.
 * @throws UnderflowException if the stack is empty.
 */
public AnyType peek( )
{
    if( isEmpty( ) )
        throw new UnderflowException( "ArrayStack top" );
    return theArray[ topOfStack ];
}
```

Stack Implementation Using an Array (Java)

```
/**
 * Internal method to extend theArray.
 */
private void doubleArray( )
{
    AnyType [ ] newArray;

    newArray = (AnyType [ ]) new Object[ theArray.length * 2 ];
    for( int i = 0; i < theArray.length; i++ )
        newArray[ i ] = theArray[ i ];
    theArray = newArray;
}
```

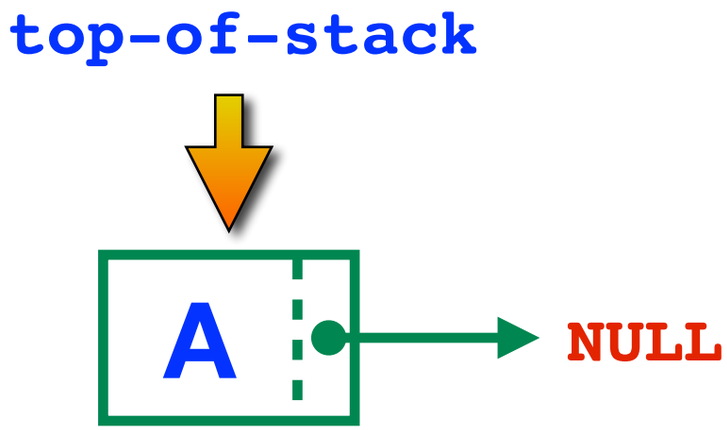
Stack Implementation Using a LinkedList

Start with Empty Stack (i.e. a null LinkedList)



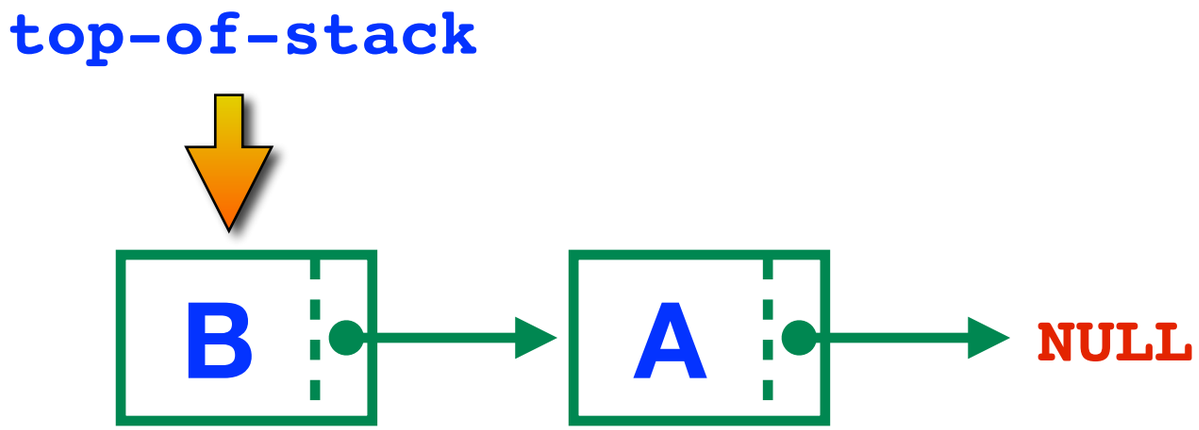
Stack Implementation Using a LinkedList

After Pushing Value: A



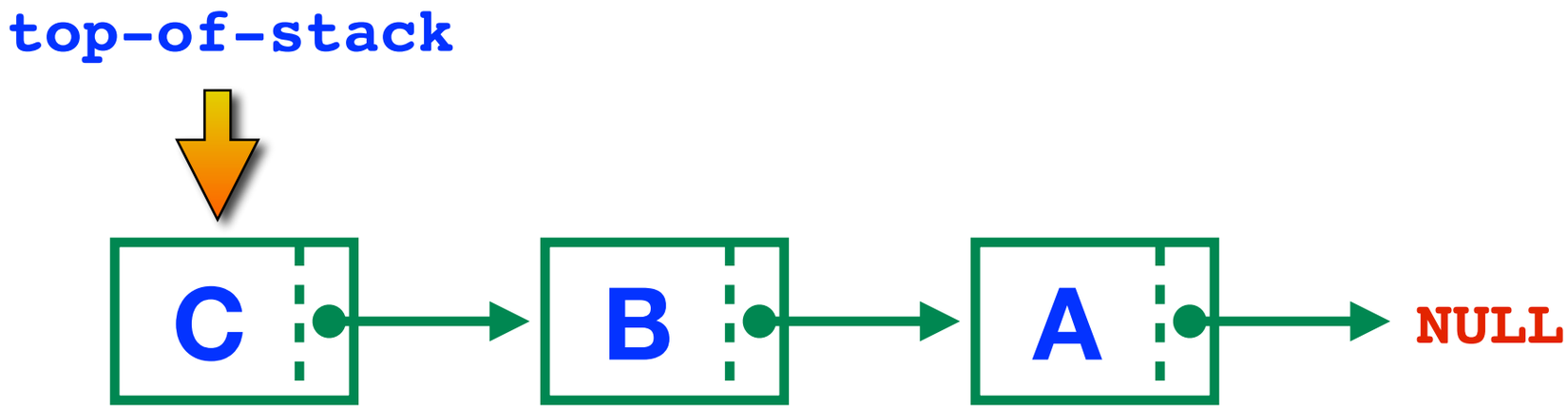
Stack Implementation Using a LinkedList

After Pushing Value: B



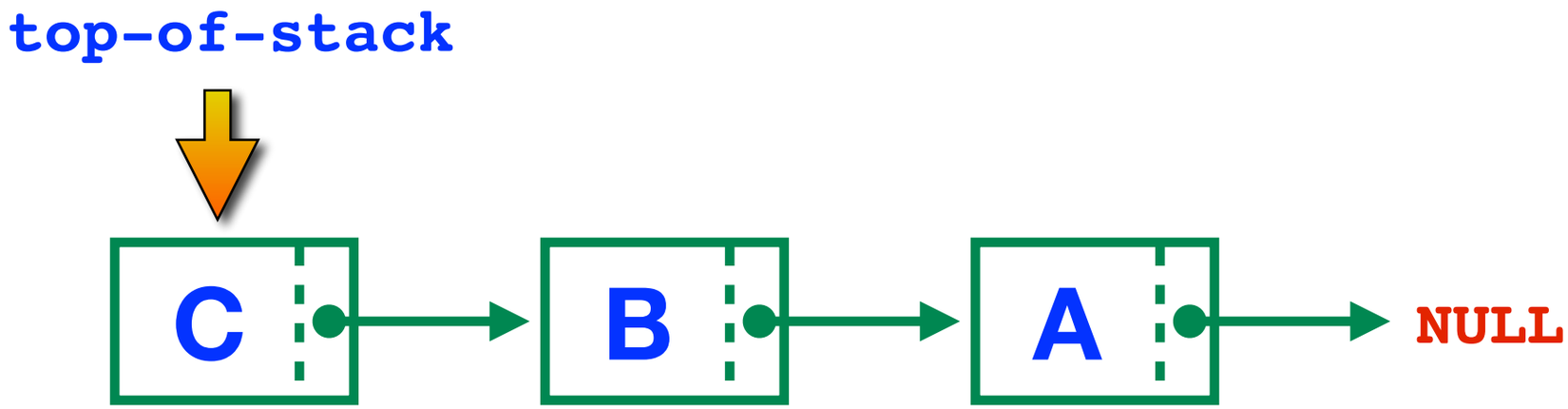
Stack Implementation Using a LinkedList

After Pushing Value: C



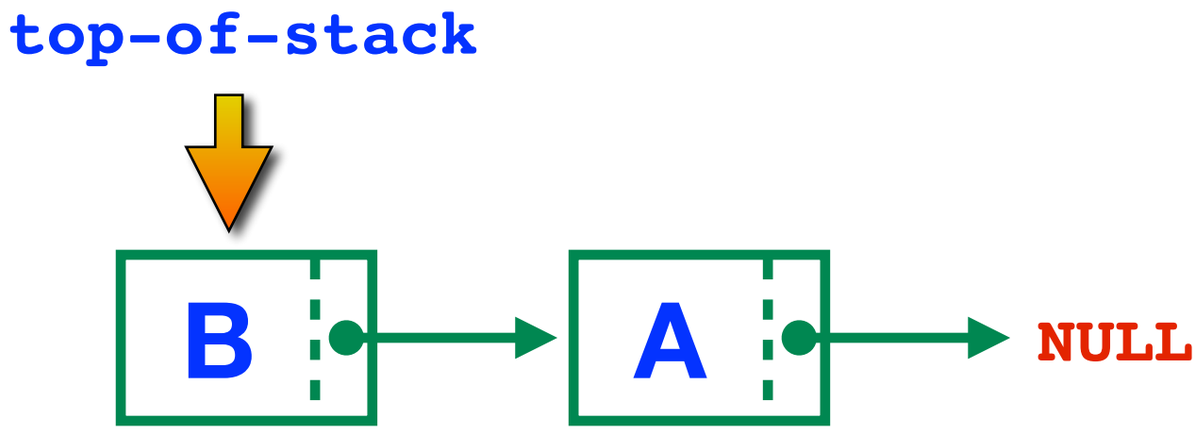
Stack Implementation Using a LinkedList

Current State of the Stack:



Stack Implementation Using a LinkedList

After Popping the Top of the Stack the Value C was removed and the top-of-stack pointer now points to B.



Stack Implementation Using a LinkedList

- **Considerations when using a LinkedList implementation**
  - **`push`**, **`pop`**, and **`peek`** operations all run in constant time ... still
  - Each element inserted into the stack requires a pointer to the data and a second pointer to the next node in the LinkedList

Stack Implementation Using a LinkedList (Java)

```
/**
 * Stack constructor
 */
public ListStack( )
{
    topOfStack = null;
}

private ListNode<AnyType> topOfStack;
```

Stack Implementation Using a LinkedList (Java)

```
/**
 * Test if the stack is logically empty.
 * @return true if empty, false otherwise.
 */
public boolean isEmpty( )
{
    return topOfStack == null;
}
```

Stack Implementation Using a LinkedList (Java)

```
/**
 * Make the stack logically empty.
 */
public void makeEmpty( )
{
    topOfStack = null;
}
```

Stack Implementation Using a LinkedList (Java)

```
/**
 * Insert a new item into the stack.
 * @param x the item to insert.
 */
public void push( AnyType x )
{
    topOfStack = new ListNode<AnyType>( x, topOfStack );
}
```

Stack Implementation Using a LinkedList (Java)

```
/**
 * Remove the most recently inserted item from the stack.
 * @throws UnderflowException if the stack is empty.
 */
public void pop( )
{
    if( isEmpty( ) )
        throw new UnderflowException( "ListStack pop" );
    topOfStack = topOfStack.next;
}
```

Stack Implementation Using a LinkedList (Java)

```
/**
 * Get the most recently inserted item in the stack.
 * Does not alter the stack.
 * @return the most recently inserted item in the stack.
 * @throws UnderflowException if the stack is empty.
 */
public AnyType peek( )
{
    if( isEmpty( ) )
        throw new UnderflowException( "ListStack top" );
    return topOfStack.element;
}
```

Example Applications for Stacks

- **Postfix expression evaluation**

2 + 3 * 4 vs 2 3 4 * +

- **Checking for balanced delimiters**
  - Balancing parenthesis/braces in a programming languages