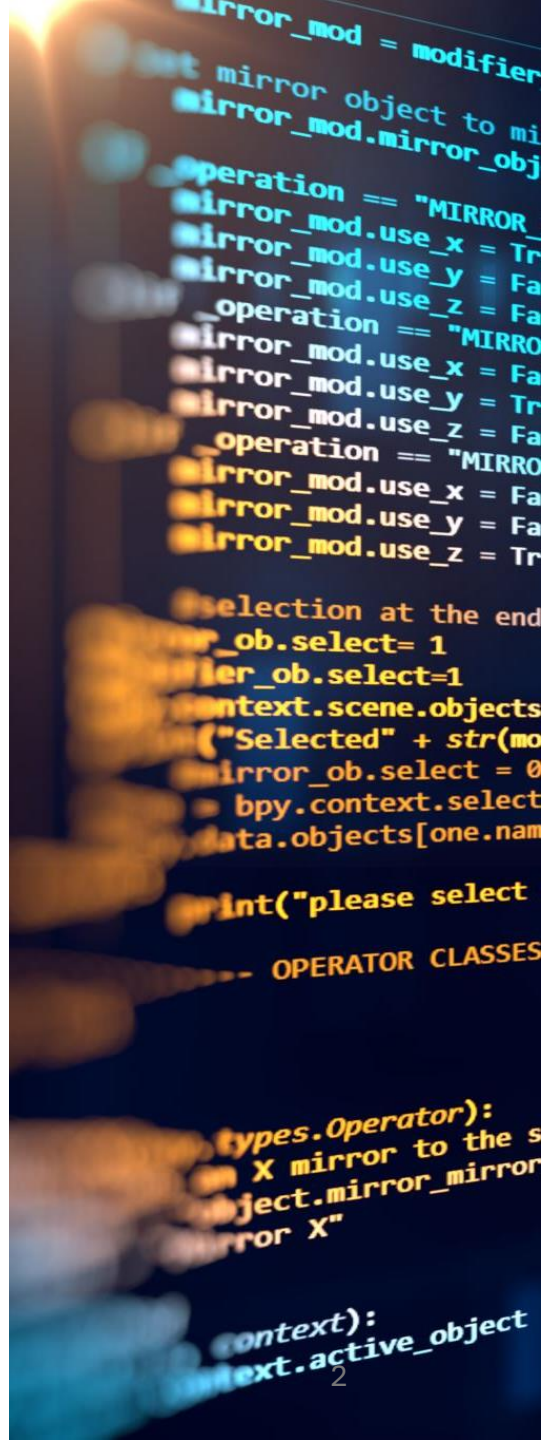
Please use the following QR code to check in and record your attendance.

CS 1037
Fundamentals of Computer Science II

Queue ADT(cont.)

Ahmed Ibrahim



Linked List Queues

- A **linked list queue** stores queue data values in a **singly linked list** and uses **two pointers**, **front and rear**, to represent the front and rear positions.
- The front pointer points to the first node, and the rear pointer points to the last node of the singly linked list.
- A linked list queue is empty if both front and rear are NULL. The queue operations are defined as follows.
  - The enqueue operation first creates a node containing the data value, inserts the node after the rear node, and updates both front and rear.
  - The dequeue operation deletes the front node (i.e., the node pointed by the front pointer) and updates the front and rear.
- The peek operation, a simple function, returns the data value in the front node.

Enqueue Algorithm

Input: front, tail, value

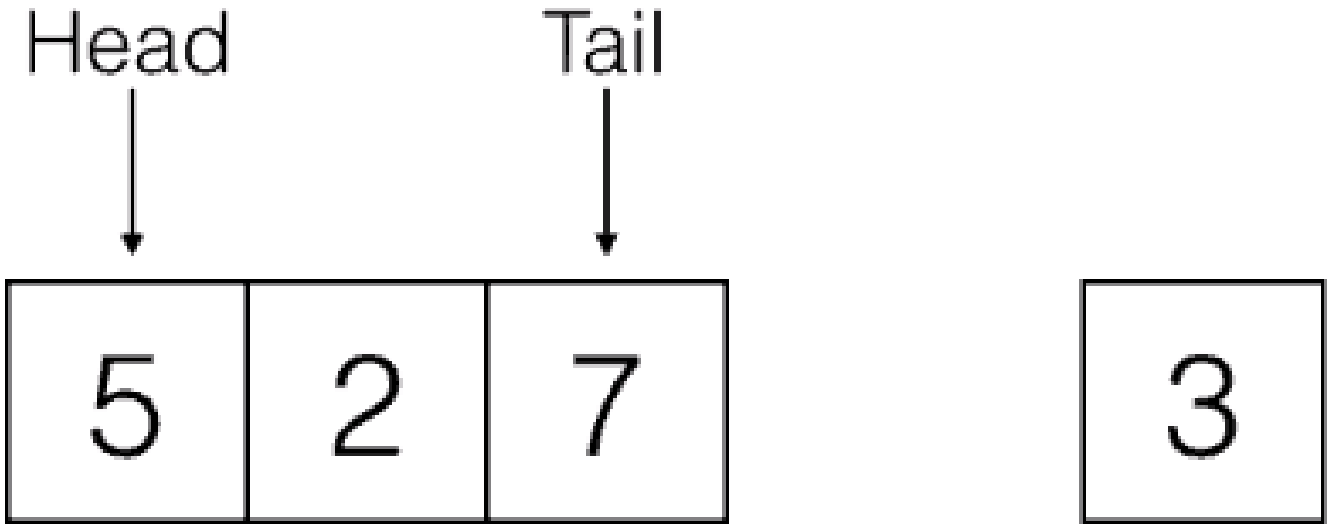
Step 1: create newNode = new_node(value).

Step 2: If front == NULL, front = newNode, tail = newNode goto step 4

Step 3: tail->next = newNode; tail = newNode;

Step 4: output front and tail

Enqueue



Source: https://thagomizer.com/blog/2016/06/13/algorithms-queues-and-stacks.html

Dequeue Algorithm

Input: front, rear

Step 1: If front == NULL

    print UNDERFLOW

    goto step 6

Step 2: ptr = front

Step 3: If front == rear

    front = NULL

    rear = NULL

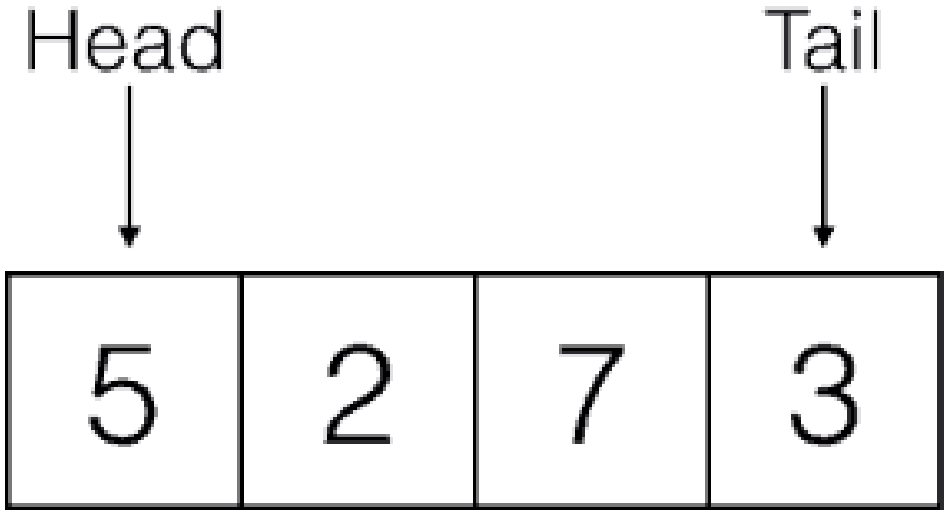
    goto step 5

Step 4: front = front->next

Step 5: free ptr

Step 6: output front and rear

Dequeue



Source: https://thagomizer.com/blog/2016/06/13/algorithms-queues-and-stacks.html

Priority Queue

Priority Queue

- A priority queue is a collection of elements in which each element is assigned a **priority**. The priority of the elements determines the order in which they will be processed.
- The rule for processing elements of a priority queue is the following:
  1. An element of a **higher priority** is processed before an element with a lower priority.
  2. Two elements of the same priority are processed on a first-come, first-served (FCFS) order.
- **A general queue can be viewed as a special priority queue using insertion time as a priority**.
- Priority queues can be implemented by either linked lists or arrays.
- **Use case**: Priority queues are used in operating systems to manage processes for running. The highest priority process will be processed first.

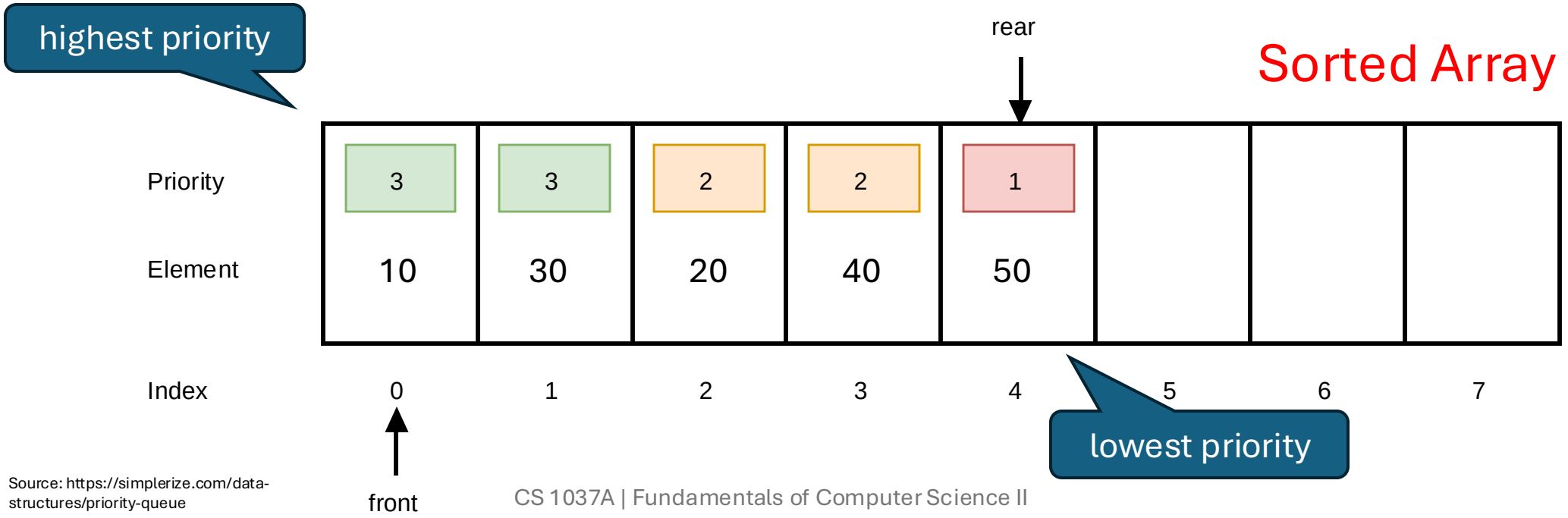
Linked List Priority Queue

- A linked list priority queue utilizes a **singly linked list** to store data values, with a pointer front that points to the first node.
- Each node in this list has three components as follows:
- The list is sorted based on priority, meaning nodes with higher priority come before those with lower priority.
- When inserting a new node, it is placed after a specific node that meets the following conditions:
  - The priority of the new node is less than or equal to the priority of the current node.
  - The new node's priority is greater than the next node's priority unless the next node is NULL.

```
typedef struct Node {
int data;
int priority;
struct Node* next;
} Node;
```

Priority Queue Sorted by Priority

- The elements are enqueued in the order 10, 20, 30, 40, and 50, each **sorted** by **priority** during the enqueue operation. This way, the highest priority element is readily accessible for quick dequeuing from the front.



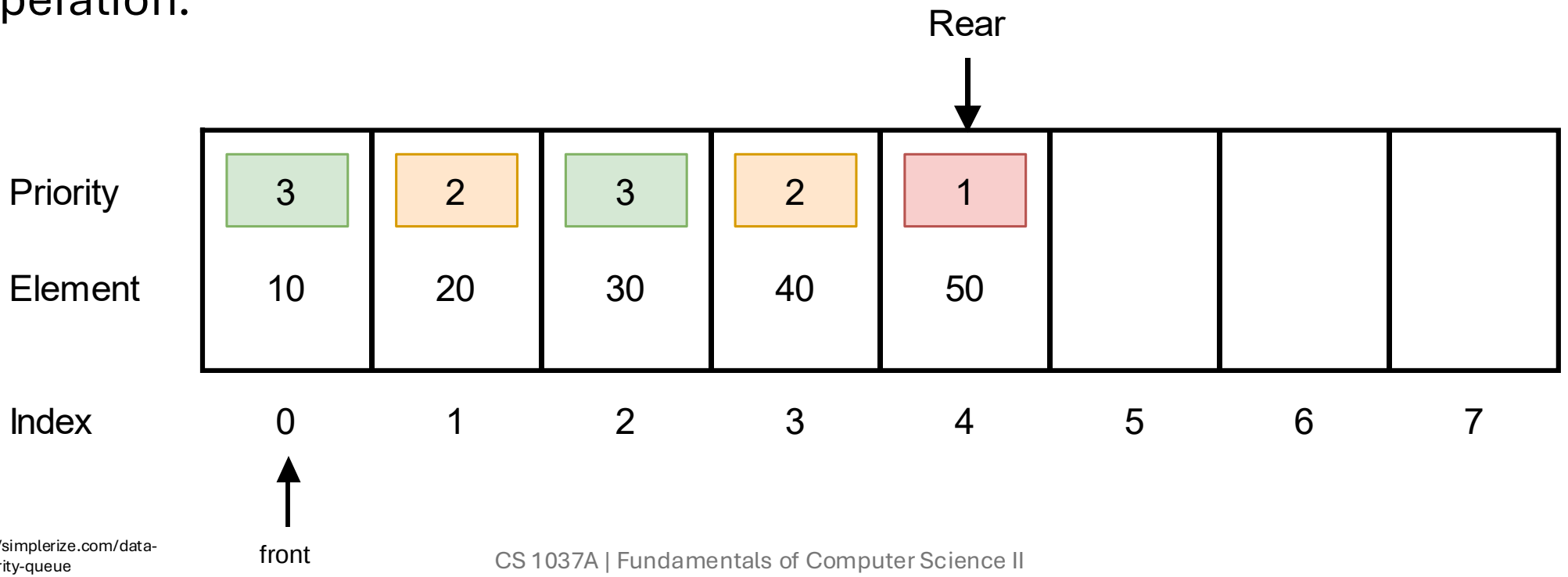
Source: https://simplerize.com/data-structures/priority-queue

Pseudocode for Enqueue in Array-Based Sorted Priority Queue

```
Function Enqueue(priorityQueue, element, priority):
  # Step 1: Create a new item with an element and priority
  newItem = (element, priority)
  # Step 2: Find the correct position to insert the new item
  position = 0
  While position < length(priorityQueue) AND
priorityQueue[position].priority >= priority:
    position = position + 1
  # Step 3: Shift elements to make space for the new item
  For i = length(priorityQueue) - 1 down to position:
    priorityQueue[i + 1] = priorityQueue[i]
  # Step 4: Insert the new item at the correct position
  priorityQueue[position] = newItem
  # Step 5: Update the rear of the queue
  rear = rear + 1
End Function
```

Array-Based Priority Queues

- The dequeue operation locates and serves the highest-priority element by searching through the entire queue, making it more costly than the enqueue operation.



Source: https://simplerize.com/data-structures/priority-queue

Pseudocode for Dequeue in Array-Based Unsorted Priority Queue

```
Function Dequeue(priorityQueue):
  # Step 1: Check if the queue is empty
  If length(priorityQueue) == 0:
    Print "Queue is empty"
    Return Null
  # Step 2: Find the highest-priority element
  highestPriorityIndex = 0
  For i = 1 to length(priorityQueue) - 1:
    If priorityQueue[i].priority > priorityQueue[highestPriorityIndex].priority:
      highestPriorityIndex = i
  # Step 3: Remove the highest-priority element
  highestPriorityElement = priorityQueue[highestPriorityIndex]
  # Step 4: Shift elements to fill the gap
  For j = highestPriorityIndex to length(priorityQueue) - 2:
    priorityQueue[j] = priorityQueue[j + 1]
  # Step 5: Remove the last element (duplicate after shifting)
  Remove the last element from priorityQueue (or decrease its length by 1)
  # Step 6: Return the dequeued element
  Return highestPriorityElement
End Function
```

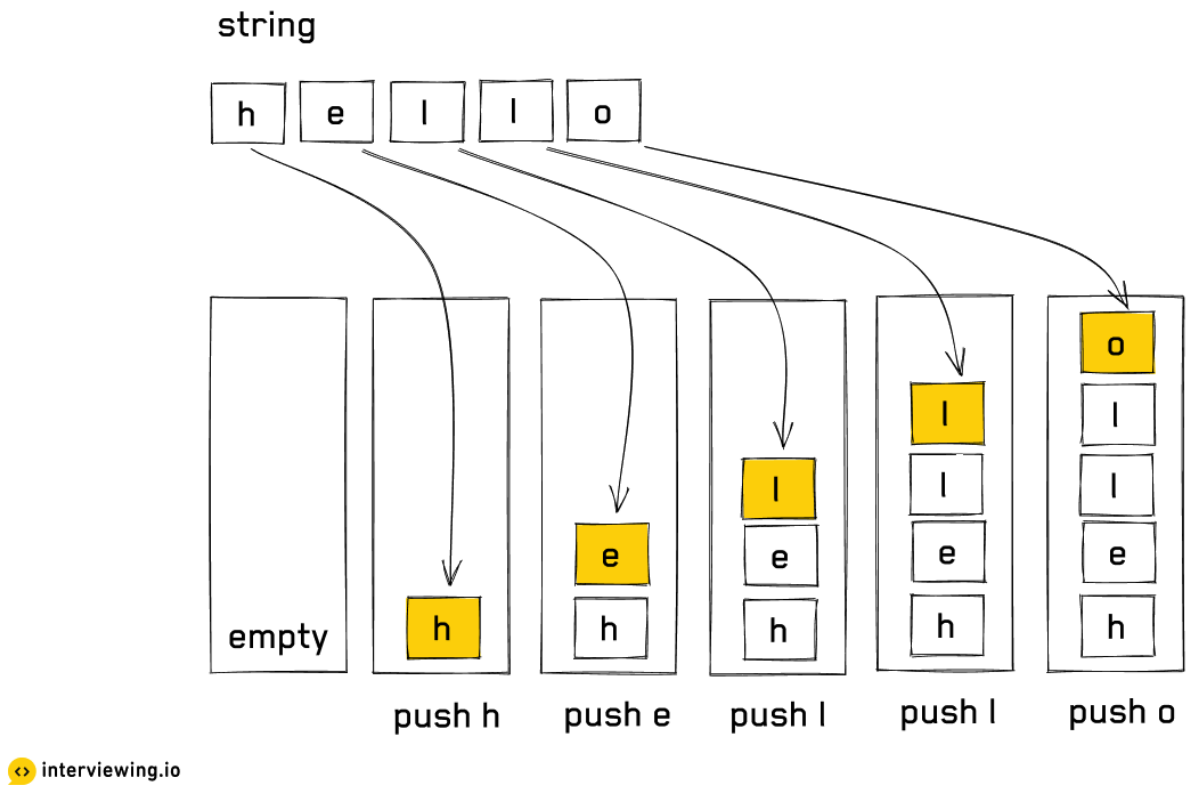
Applications Using ADTs

Reverse A String

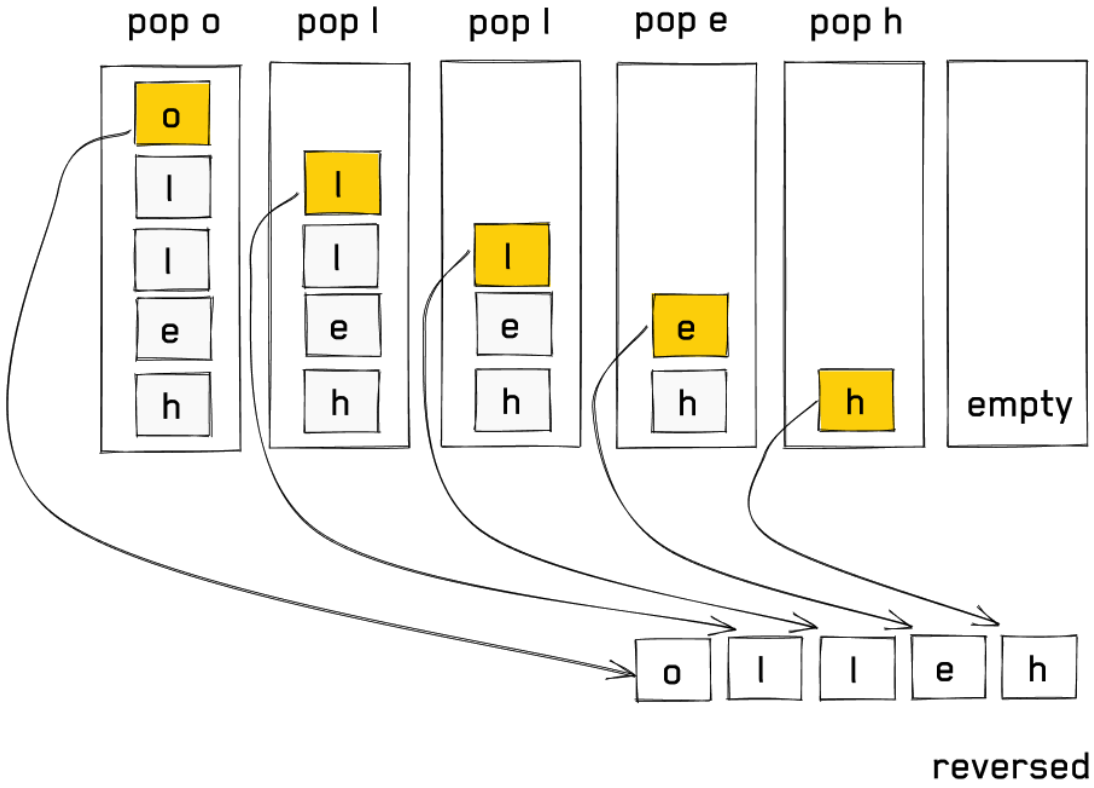
Imagine you need to reverse a string like "hello" using the Abstract Data Types (ADT) we've discussed so far.

- What approach would you follow to achieve this?
- What steps would you take to accomplish this task?

Reverse A String /1



Reverse A String /2



interviewing.io

Pseudocode to Reverse a String Using a Stack ADT

```
Function ReverseString(inputString):
    # Step 1: Initialize an empty stack
    stack = EmptyStack()
    # Step 2: Push each character of the input string onto the stack
    For each character in inputString:
        stack.Push(character)
    # Step 3: Initialize an empty string to store the reversed result
    reversedString = ""
    # Step 4: Pop characters from the stack and append them to
the reversed string
    While not stack.IsEmpty():
        reversedString = reversedString + stack.Pop()
    # Step 5: Return the reversed string
    Return reversedString
End Function
```

Question!

Consider the given pseudocode to reverse a string using a stack, and answer the following:

a) Suppose we want to reverse each word in a sentence individually (e.g., "Hello World" becomes "World Hello"). Modify the pseudocode to handle this case.

b) Discuss any additional complexity or edge cases that may arise in this modified version, such as handling multiple spaces or special characters.

Another Stack Implementation

- Infix and Postfix notations are two different but equivalent notations for writing algebraic expressions.

Infix	Postfix	Notes
A * B + C / D	A B * C D / +	multiply A and B, divide C by D, add the results
A * (B + C) / D	A B C + * D /	add B and C, multiply by A, divide by D
A * (B + C / D)	A B C D / + *	divide C by D, add B, multiply by A

Postfix Notation

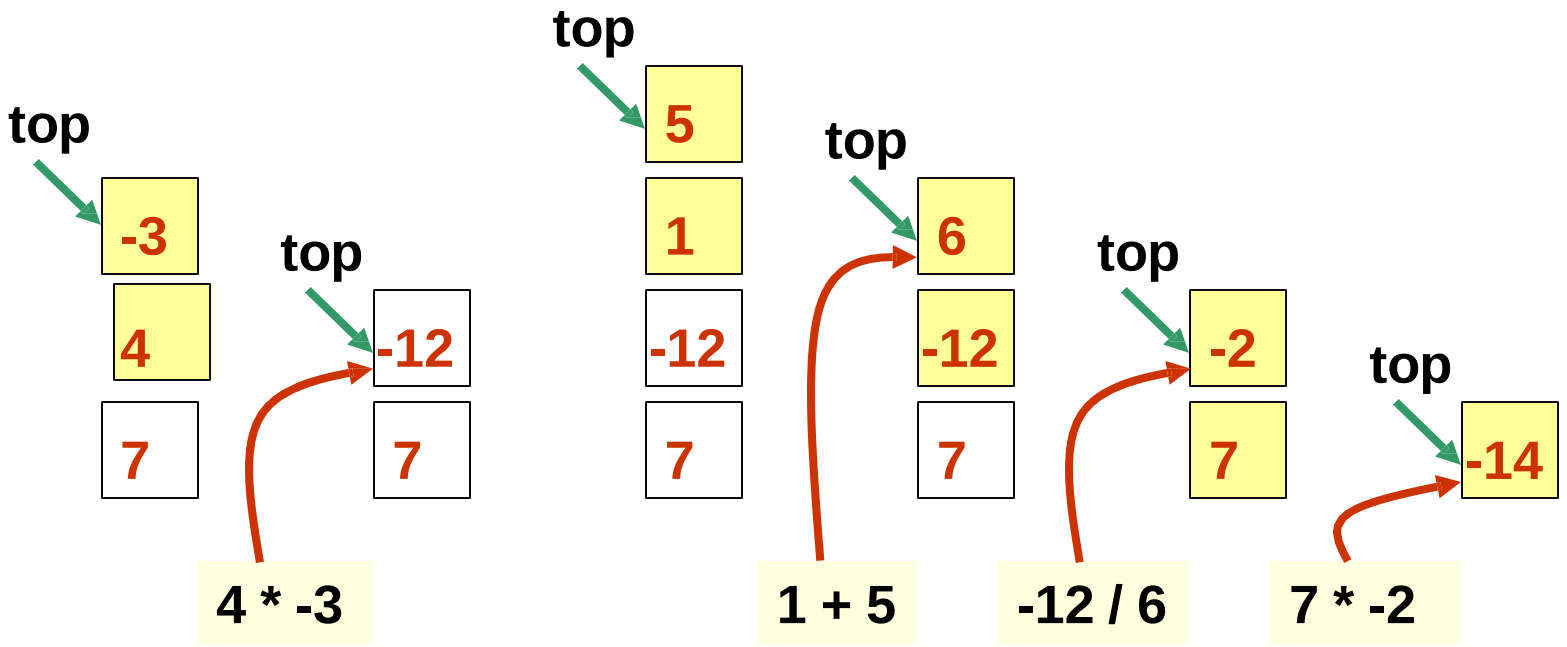
- A Polish mathematician gave Postfix notation. His aim was to develop a **parenthesis-free** prefix notation (also known as Polish notation).
- In postfix notation, the operator is placed after the operands. For example, if an expression is written as A+B in **infix** notation, the same expression can be written as AB+ in **postfix** notation.
- The order of postfix expression evaluation is always from **left** to **right**.

Evaluating A Postfix Expression

- Algorithm to evaluate a postfix expression:
  - Scan from left to right, determining if the next term is an operator or operand
  - If it is an operand, push it on the stack
  - If it is an operator, pop the stack twice to get the two operands, perform the operation, and push the result back onto the stack.
- Try the algorithm in the following example; ultimately, the stack will contain a single value.

Using a Stack to Evaluate a Postfix Expression

Evaluation of 7 4 -3 * 1 5 + / *



The result is the only item on the stack at the end of the evaluation.

Pseudocode to Evaluate a Postfix Expression Using a Stack ADT

```
Function EvaluatePostfix(expression):
  # Step 1: Initialize an empty stack
  stack = EmptyStack()
  # Step 2: Scan each term in the expression from left to right
  For each term in expression:
    # Step 3: Check if the term is an operand
    If term is an operand:
          stack.Push(term) # Push the operand onto the stack
    # Step 4: If the term is an operator
    Else if term is an operator:
      # Pop the stack twice to get the two operands
      operand2 = stack.Pop()
      operand1 = stack.Pop()
      # Perform the operation on the two operands
      result = PerformOperation(operand1, operand2, term)
      # Push the result back onto the stack
      stack.Push(result)
  # Step 5: After processing all terms, the stack will contain the final result
  finalResult = stack.Pop()
  Return finalResult # Step 6: Return the final result
End Function
```

Stack Applications: Undo and Redo Functionality

- Two stacks can provide undo and redo functionality for text editors or graphic design software applications.
- **Approach**:
  - Use one stack (undoStack) to store each action performed by the user.
  - When an action is undone, pop from undoStack and push it onto a redoStack.
  - When a redo is performed, pop from redoStack and push it back to undoStack.
- Many interactive applications, including text editors and drawing tools, use this approach to allow users to undo and redo their actions.

Queue Applications: Managing Multi-Level Cache in Web Servers

- In cache management, two queues can be used to maintain different cache levels (e.g., frequently accessed data vs. less frequently accessed data).
  - Use one queue to store frequently accessed items and another for less frequently accessed items.
  - Move items between the queues based on access patterns.
- This can optimize web servers, databases, or applications requiring multi-level caching for performance improvement.

Task Scheduling with Completion Tracking

- Task Scheduling with Completion Tracking is a system that uses two ADTs—a queue and a stack—to efficiently manage and monitor tasks:
- Use a queue to manage incoming tasks.
- As each task is completed, push it onto a stack for tracking completion.
- This allows for easy access to recently completed tasks if needed.
- Useful in workflow systems or project management tools to track completed tasks and allow quick access to the latest completions.


thank you