Linear Data Structures

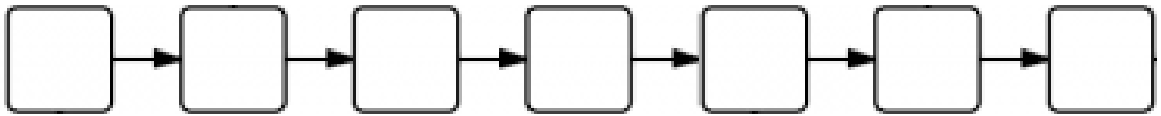
Stacks and Queues

Data Structures

- a way to store and organize data in order to facilitate access and modifications
- Categorized into:
  - Linear
  - Non-linear
- no single data structure works well for all purposes

Linear Data Structures

What is a linear data structure?
What makes a data structure “linear”?



Examples:

- Arrays
- Linked Lists
- Stacks
- Queues

Arrays

- Mainly used to store *homogenous* data
- Data is stored in consecutive memory locations
- Data is referenced by an *index*, and returns a *value*

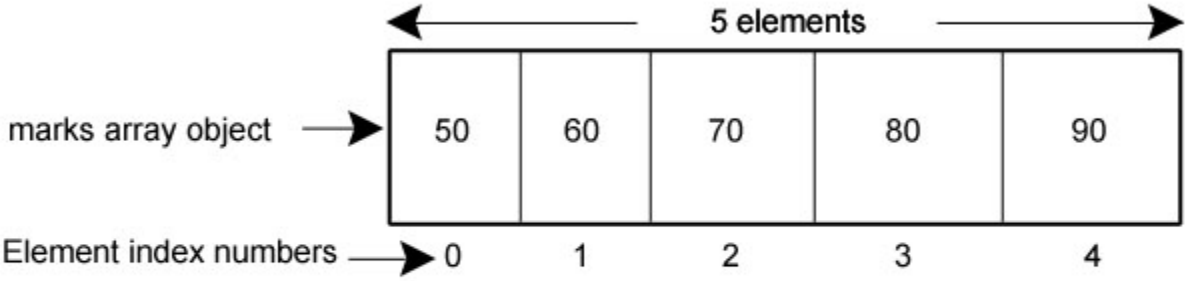
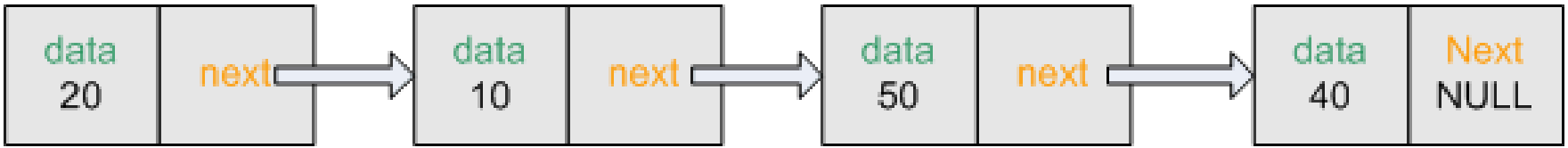


Figure : marks array with 5 elements
(after assigning values to elements)

Date	Client number	Payment
03.11.2026	24	YES
10.23.2018	30	YES
03.21.2021	6	YES
03.08.2023	24	YES
04.05.2016	29	YES
03.22.2018	10	YES
07.27.2021	30	YES
09.25.2025	5	YES
02.04.2016	23	NO
09.11.2019	20	NO
01.25.2020	28	YES
07.23.2025	18	NO
06.01.2011	25	YES
12.03.2017	14	YES

Linked Lists

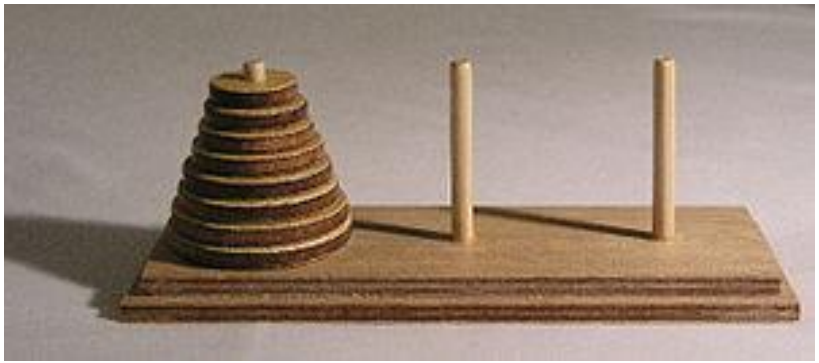
- Still mainly used to store *homogenous* data
- Data is **not** stored in consecutive memory locations
- The list is **searched** until the index/element is found



Linked list

Other Linear Data Structures

Using Arrays and Linked Lists, we can create other linear data structures to model real-life objects/happenings:



Other Linear Data Structures

Using Arrays and Linked Lists, we can create other linear data structures to model real-life objects/happenings:

Image retrieved from http://registerguard.com/rg/news/local/35871206-75/eclipse-related-traffic-piles-up-in-eastern-oregon.html.csp

Abstract Data Types

- Stacks
  - https://www.youtube.com/watch?v=CgFVgp_VCN8&feature=share
- Queues
  - https://www.youtube.com/watch?v=ligWuGbhUMY&feature=share

Stacks

- **LIFO (Last In First Out)**
- Only the top element is accessible
- Insertions are made over the top element
  - The top element is replaced by the new insertion
- Deletions remove the top element

Stack Operations

- **CREATE**(STACK) – creates an empty stack
- **PUSH**(STACK, ITEM) – inserts an element item into the stack (INSERT operation)
- **POP**(STACK) – removes and then returns the top element of the stack (DELETE operation)
- **TOP**(STACK)/**PEEK**(STACK) – returns the top element of the stack

Stack Operations

- **isEmpty**(STACK) – determines whether the stack is empty or not

- **isFull**(STACK) – determines whether the stack is full or not

https://visualgo.net/en/stack

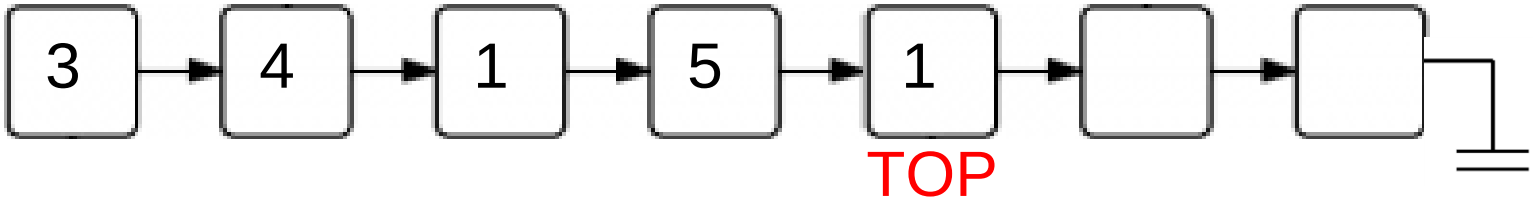
Stack: Possible Errors

- **Underflow** – occurs when an empty stack is popped
- **Overflow** – occurs when pushing an element to a fully-filled stack (i.e. TOP > n)

Stack Representations

1. 1D Array

1. Singly Linked List



Examples

Tower of Hanoi

Parentheses/Brackets/Braces matching

Queues

- **FIFO (First In First Out)**
- Only the front element is accessible
- Insertions are done from the back of the queue
- Deletions are done on the front

Queue Operations

- **CREATE**(QUEUE) – creates an empty queue
- **ENQUEUE**(QUEUE, ITEM) – inserts an element into the queue (INSERT operation)
- **DEQUEUE**(QUEUE) – removes and then returns the head of the queue (DELETE operation)

https://visualgo.net/en/queue

Queue Operations

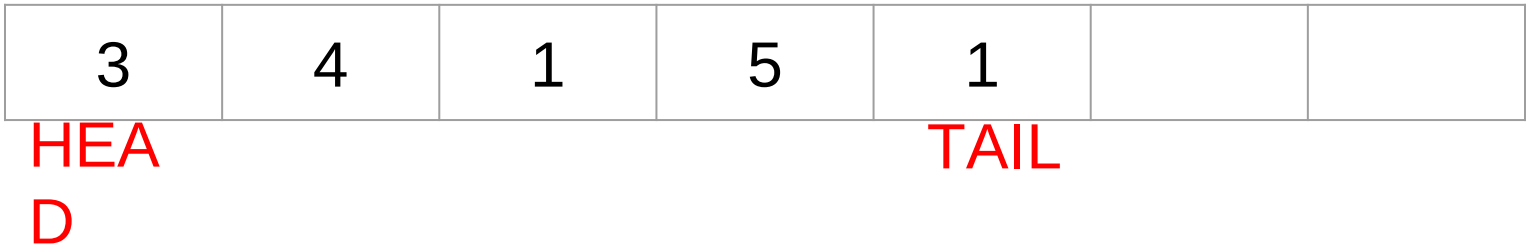
- **QUEUE_HEAD**(QUEUE)/**FRONT**(QUEUE) – determines the element at the head of the queue
- **QUEUE_TAIL**(QUEUE)/**BACK**(QUEUE) – determines the element at the tail of the queue
- **isQEmpty**(QUEUE) – determines if the queue is empty or not
- **isQFull**(QUEUE) – determines if the queue is full or not

Queue: Possible Errors

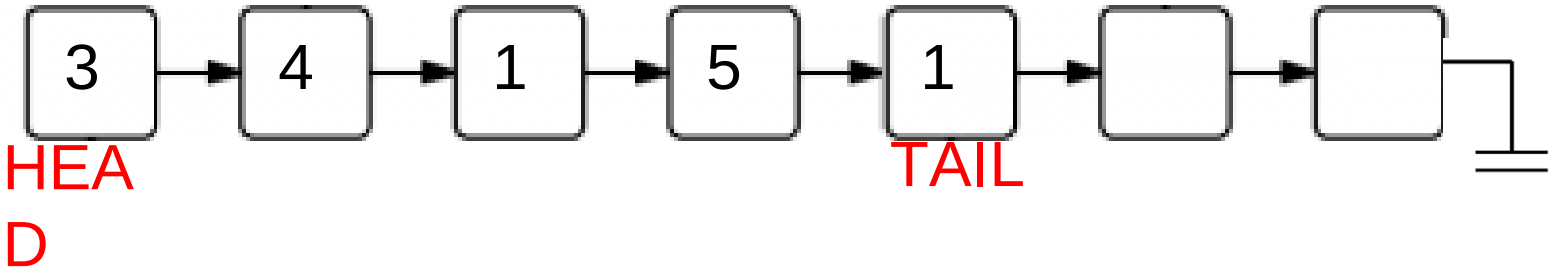
- **Underflow** – occurs when an empty queue is being dequeued
- **Overflow** – occurs when enqueueing an element to a queue that is full

Queue Representations

1. 1D Array



1. Singly Linked List



Stacks and Queues Example

The **Shunting-Yard Algorithm** is an algorithm that uses stacks and queues to convert a mathematical expression from **infix** to **postfix** form and evaluate later on.

Operator	Priority	Associativity
()	1 (highest)	Inner to Outer
^	2	**Right to Left**
*, /	3	Left to Right
+, -	4 (lowest)	Left to Right

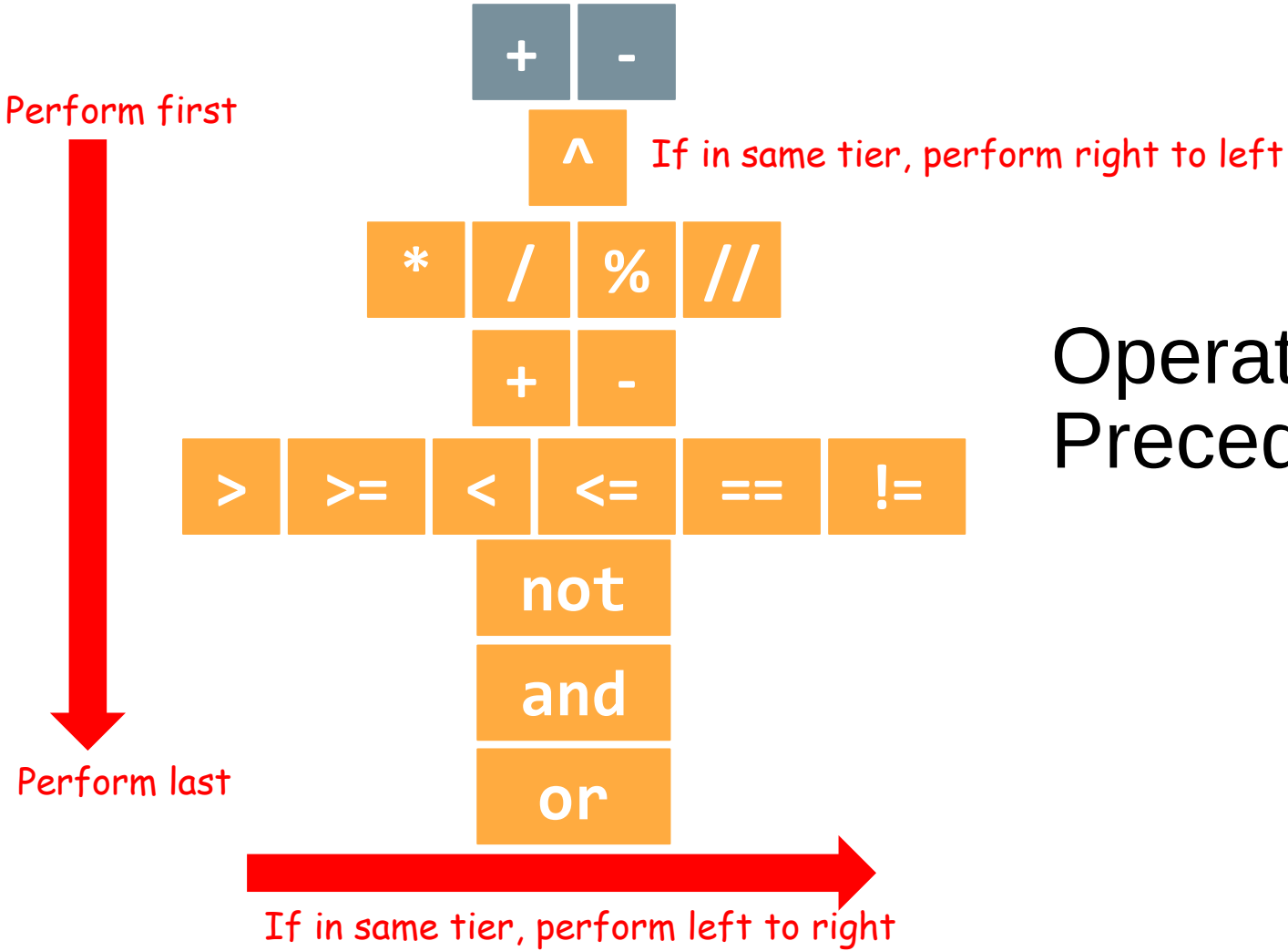
https://www.youtube.com/watch?v=y_snKkv0gWc

Stacks and Queues Example

Evaluating mathematical expression in **prefix** form using stacks and queues.

Operator	Priority	Associativity
()	1 (highest)	Inner to Outer
^	2	**Right to Left**
*, /	3	Left to Right
+, -	4 (lowest)	Left to Right

Operator Precedence



Mathematical expressions

Three notations:

1. Infix

   <operand> **<operator>** <operand> a+b

2. Prefix

   <operator> <operand> <operand> +ab

3. Postfix

   <operand> <operand> **<operator>** ab+

Mathematical expressions

https://www.youtube.com/watch?v=y_snKkv0gWc
https://www.youtube.com/watch?v=MeRb_1bddWg

Mathematical expressions

Conversion from Prefix or Postfix to other notations:

1. Format
   - Prefix: **<operator>** <operand> <operand>
   - Postfix: <operand> <operand> **<operator>**
2. Direction of evaluation
   - Prefix: Right to Left
   - Postfix: Left to Right

Mathematical expressions

Conversion from Prefix or Postfix to Infix:

1. Look for the first operator from the right (i.e. if Prefix) or the left (i.e. if Postfix). That operator is for the last 2 operands.
2. Convert to Infix
3. Look at your current Infix. If there are operators lower than the current, enclose those operators and their operands inside parentheses.
4. Go back to step 1 until everything is converted.

Mathematical expressions

Conversion from Prefix to Postfix and vice versa:

1. Look for the first operator from the right (i.e. if Prefix) or the left (i.e. if Postfix). That operator is for the last 2 operands.
2. Convert to Prefix or Postfix
3. Go back to step 1 until everything is converted.

Mathematical expressions

Remember:

1. The order of the operands will always be the same.
2. The order and position of the operators will change depending on the notation.

Questions?