Computer Fundamentals

Priti Srinivas Sajja
Professor
Department of Computer Science
Sardar Patel University

Visit pritisajja.info for details

Unit 4: Data Structure

- **Introduction**
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

- **Name: Dr. Priti Srinivas Sajja**
- **Communication:**
  - **Email :** priti@pritisajja.info
  - **Mobile :** +91 9824926020
  - URL :http://pritisajja.info

- ***Academic qualifications*** **: Ph. D in Computer Science**
- ***Thesis title***: **Knowledge-Based Systems for Socio-**
- **Economic Rural Development (2000)**
- ***Subject area of specialization*** **: Artificial Intelligence**
- ***Publications*** **: 216 in Books, Book Chapters, Journals and in Proceedings of International and National Conferences**

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Unit 4: Data Structures

- Primitive and composite data types
- Arrays, stacks, queues, linked lists
- Binary trees, B-trees
- Hashing techniques
- Linear Search, Binary Search
- Bubble Sort

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

- **Data:** Row observations and values

- **Structure:** Way of organizing the values, so that it is easier to use

- The data structure is defined as a **way of organizing data** in such a way so that data can be used **efficiently (Space and Time)**.

Unit 4: Data Structure

- Introduction
- **Need of DT**
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Need For Data Structure

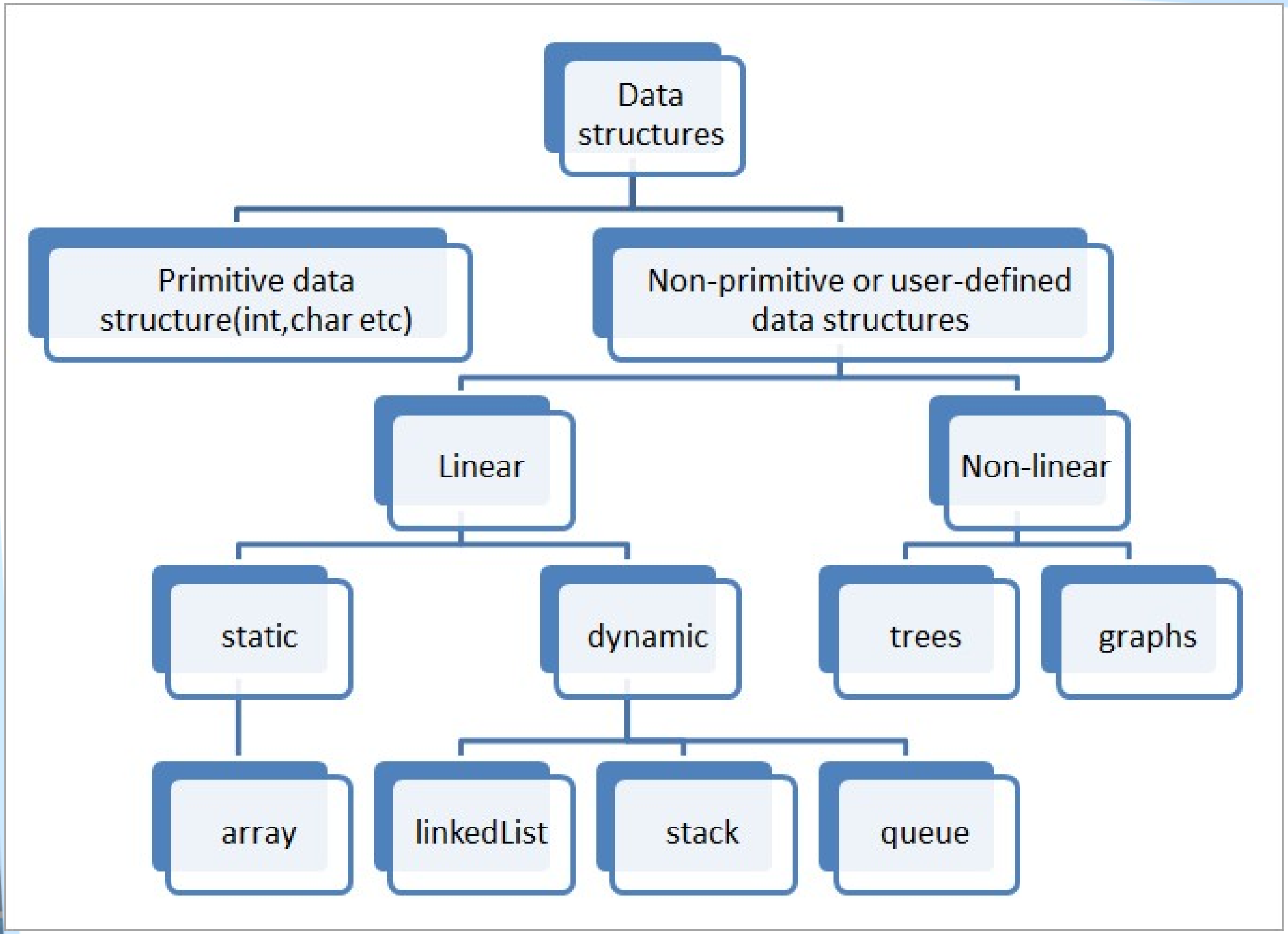
- **Searching Large amounts of Data**: To retrieve required data efficiently from the large amount of data generated and stored
- **Speed of Processing:** Searching and retrieving data from the well organized bunch takes less time and less effort.
- **Concurrent Requests:** Many and simultaneous requests can be easily handled.

Characteristics of a Data Structure

- **Correctness** – correctly implemented.
- **Time Complexity** – Running time or the execution time of operations of data structure must be as small as possible.
- **Space Complexity** – Memory usage of a data structure operation should be as little as possible.

Unit 4: Data Structure

- Introduction
- **Type of DT**
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort



Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Operations on Data Structure

- Define or create structure
- Add an element
- Delete an element
- Traverse / Display
- Sort the list of elements
- Search for a data element
- Merging and spiting
- Delete an element or delete complete structure

Unit 4: Data Structure

Primitive Data Structures

- Primitive data structures are **basic structures** and are **directly operated** upon by machine instructions.
- Primitive data structures have different representations on different computers.
- **Integers, floats, character and pointers** are examples of primitive data structures.
- These data types are available in most programming languages as **built in type**.

- Introduction
- Types of DT
- Array
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Non Primitive Data Structures

- These are **more sophisticated** data structures.
- These are **derived from primitive data** structures.
- A Non-primitive data type is further divided into **Linear and Non-Linear** data structure
- **Linear→ Array, Linked list, Stack and Queue**
- **Non linear→ Tree and Graph**

Unit 4: Data Structure

Linear Data Structures

- A data structure is said to be Linear, if its **elements are connected in linear fashion** by means of logically or in sequence memory locations.
- There are two ways to represent a linear data structure in memory,
  - **Static memory allocation**
  - **Dynamic memory allocation**

Unit 4: Data Structure

Difference between Linear and Non Linear Data Structure

	Linear Data Structure	Non-Linear Data Structure
1	Every item is related to its previous and next time.	Every item is attached with many other items.
2	Data is arranged in linear sequence.	Data is not arranged in sequence.
3	Data items can be traversed in a single run.	Data cannot be traversed in a single run.
4	Eg. Array, Stacks, linked list, queue.	Eg. tree, graph.
5	Implementation is easy.	Implementation is difficult.

Unit 4: Data Structure

- Introduction
- Types of DT
- **Array**
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Array

- Group of data with **same type and same size stored adjacent** to each other
- Arrays are always stored in **consecutive** memory locations.
- It is a **linear data structure** with known number of elements.
- Each memory location stores one fixed-length data item
- Used in all programming languages
- Can be used to **create other data structures** such as stacks and queues.
- It can be one dimensional or many dimensional.

Unit 4: Data Structure

One Dimensional Array of Integers

A[0]	A[1]	A[2]	A[3]	A[4]
10	20	30	40	50

- Here A is the name of an array.
- The value in bracket are called index.
- To refer 3^{rd} item in array A, A[2] is used.
- Total number of elements are 5.
- All are integers.
- If you know the starting address of A and size of the data, you can know address of any element in the array by knowing the index value.

- Introduction
- Types of DT
- **Array**
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Two Dimensional Array of Integers

- A[Row, Col]

Columns→ Row	0	1	2	3	4
0	A[0,0]	A[0, 1]	A[0, 2]	A[0, 3]	A[0, 4]
1	A[1,0]	A[1, 1]	A[1, 2]	A[1, 3]	A[1, 4]
2	A[2,0]	A[2, 1]	A[2, 2]	A[2, 3]	A[2, 4]
3	A[3,0]	A[3, 1]	A[3, 2]	A[3, 3]	A[3, 4]
4	A[4,0]	A[4, 1]	A[4, 2]	A[4, 3]	A[4, 4]

Two Dimensional Array of Integers

- A[Row, Col]

Columns→ Row	0	1	2	3	4
0	11	12	13	14	15
1	21	22	23	24	25
2	31	32	33	34	35
3	41	42	43	44	45
4	51	52	53	54	55

Two Dimensional Array of Integers

- Marks[Row, Col]

Columns→ Row	Sub 1	Sub 2	Sub 3	Sub 4	Sub 5
1st student	67	45	68	79	56
2nd student	66	77	85	43	78
3rd student	56	78	98	34	55
4th student	23	45	56	67	77
5th student	55	44	66	77	65

Unit 4: Data Structure

- Introduction
- Types of DT
- **Array**
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Examples of one dimensional array

An array of five characters called C

- C[0]= ‘A’
- C[1]=‘B’
- C[2]=‘C’
- C[3]=‘D’
- C[4]=‘E’

C[0]	C[1]	C[2]	C[3]	C[4]
‘A’	‘B’	‘C’	‘D’	‘E’

Introduction
Types of DT
Array
Stack
Queue
Linked Lists
Tree and Graph
Hashing
Search & Sort

Examples of one dimensional array

An array of five integer numbers called NUM

NUM[0]	NUM[1]	NUM[2]	NUM[3]	NUM[4]
44	55	66	77	88

- NUM[0]= 44 NUM[1]=55
- NUM[2]=66 NUM[3]=77
- NUM[4]=88

Unit 4: Data Structure

- Introduction
- Types of DT
- **Array**
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Examples of one dimensional array

An array of five integer numbers called NUM

NUM[0]	NUM[1]	NUM[2]	NUM[3]	NUM[4]
44	55	66	77	88

- **Find out NUM[0] + NUM[1]**

  = 44+55 =99
- **Find out NUM[4]- Num[3]**

  = 88-77=11
- **Find out yourself…**
- NUM[0] + NUM[1]+ NUM[2] +NUM[3] +NUM[4]

Unit 4: Data Structure

- Introduction
- Types of DT
- **Array**
- Stack
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Examples of one dimensional array

NUM[0]	NUM[1]	NUM[2]	NUM[3]	NUM[4]
44	55	66	77	88

- **Calculate total of all five integers from array NUM.**
  - = NUM[0] + NUM[1] + NUM[2] + NUM[3] + NUM[4]
  - = 44+55+66+77+88 = 330
- **Calculate average of all five integers from array NUM.**
  - = (NUM[0] + NUM[1] + NUM[2] + NUM[3] + NUM[4])/5
  - = (44+55+66+77+88) /5 = 330/5 = 66

Unit 4: Data Structure

Examples of one dimensional array

An array of five real numbers called NUM2

NUM2[0]	NUM2[1]	NUM2[2]	NUM2[3]	NUM2[4]
44.0	55.2	66.3	77.6	88.9

- NUM2[0]= 44.0
- NUM2[1]=55.2
- NUM2[2]=66.3
- NUM2[3]=77.6
- NUM2[4]=88.9

- You can calculate total, average, maximum number, and minimum number fro the array.

Two Dimensional Array of Characters called Names

	0	1	2	3	4
0	‘K’	‘i’	‘r’	‘t’	‘i’
1	‘P’	‘r’	‘i’	‘t’	‘i’
2	‘D’	‘i’	‘p’	‘t’	‘i’
3	‘S’	‘w’	‘a’	‘t’	‘i’
4	‘S’	‘t’	‘u’	‘t’	‘i’

- Names[0, 0] = ‘K’
- Names [1, 0]= ‘P’

Unit 4: Data Structure

- Defining An Array in Various Programming Languages

```
Array Declaration In Different Languages:

JAVA                                  long arr [] = new long [5];

 C                                    long arr[5];

Python                                arr = [None] * 5

JavaScript                            var arr = [];
```

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- **Stack**
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Stack

- Named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

- **First in Last out** Structure
- Only **top element** can be accessed.
- For example, we can place or remove a card or plate from the top of the stack only.

Unit 4: Data Structure

- Stack A stack can be implemented by means of Array, Structure, Pointer, and Linked List.
- Stack can either be a fixed size one or it may have a sense of dynamic resizing.

Basic Operations

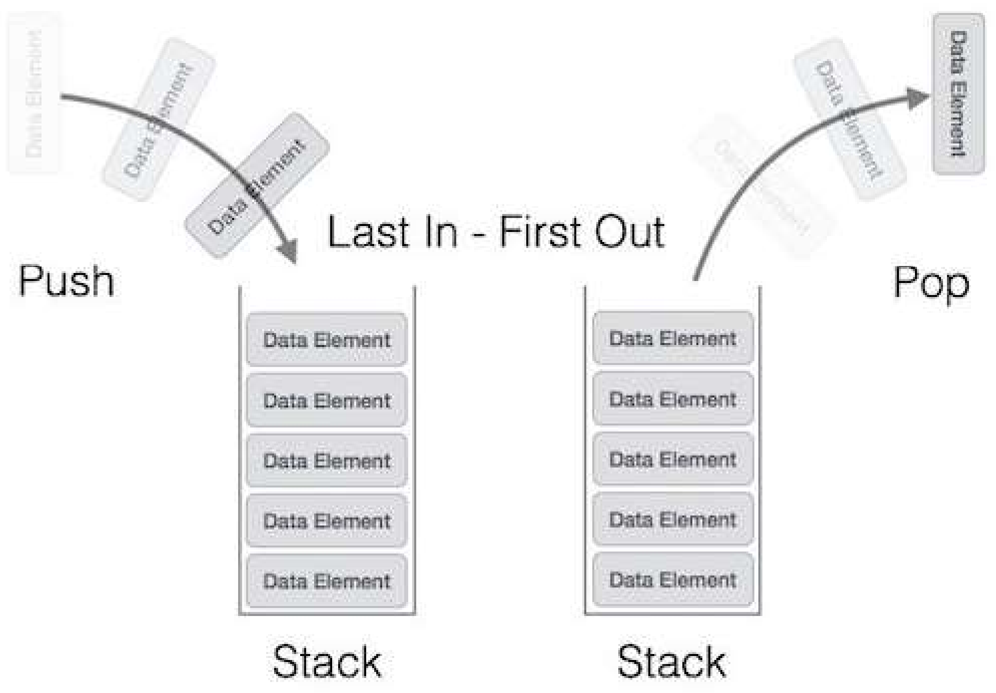
- **push()** – Pushing (storing) an element on the stack.
- **pop()** – Removing (accessing) an element from the stack.

Other Operations

- **peek()** – get the top data element of the stack, without removing it.
- **isFull()** – check if stack is full.
- **isEmpty()** – check if stack is empty.

Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- **Stack**
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort



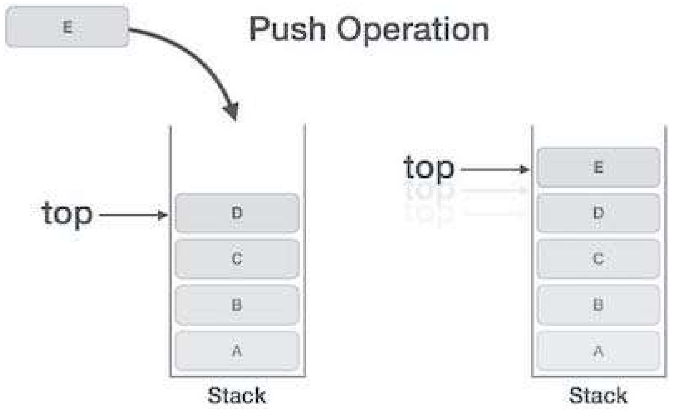
Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- **Stack**
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Push Operation

(Putting a new data element into the stack)

- **Step 1** – Checks if the stack is full.
- **Step 2** – If the stack is full, produces an error and exit.
- **Step 3** – If the stack is not full, increments **top** to point next empty space.
- **Step 4** – Adds data element to the stack location, where top is pointing.
- **Step 5** – Returns success.



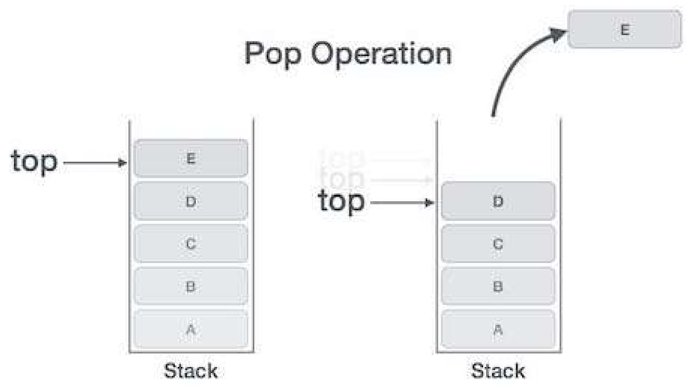
Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- **Stack**
- Queue
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

Pop Operation

(Taking off the top data element from the stack)

- **Step 1** – Checks if the stack is empty.
- **Step 2** – If the stack is empty, produces an error and exit.
- **Step 3** – If the stack is not empty, accesses the data element at which **top** is pointing.
- **Step 4** – Decreases the value of top by 1.
- **Step 5** – Returns success.



Unit 4: Data Structure

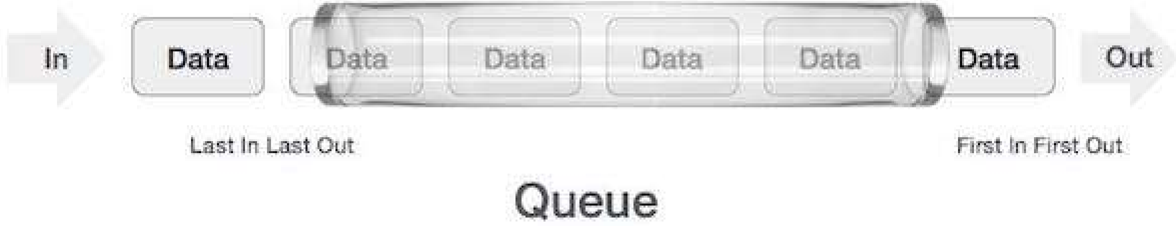
Uses of Stack

- Parsing expression (infix, prefix and postfix conversion)
- Recursion
- Flow of control and function call
- Back tracking procedures and games

Unit 4: Data Structure

Queue

- First in first out



Unit 4: Data Structure

- Introduction
- Types of DT
- Array
- Stack
- **Queue**
- Linked Lists
- Tree and Graph
- Hashing
- Search & Sort

▪ A queue can be implemented by means of Array, Structure, Pointer, and Linked List.

▪ Stack can either be a fixed size one or it may have a sense of dynamic resizing.

Basic Operations

▪ **Insert ()** – always at the end

▪ **Delete ()** – always from the front

Other Operations

▪ **peek()** – get the data element from the queue, without removing it.

▪ **isFull()** – check if stack is full.

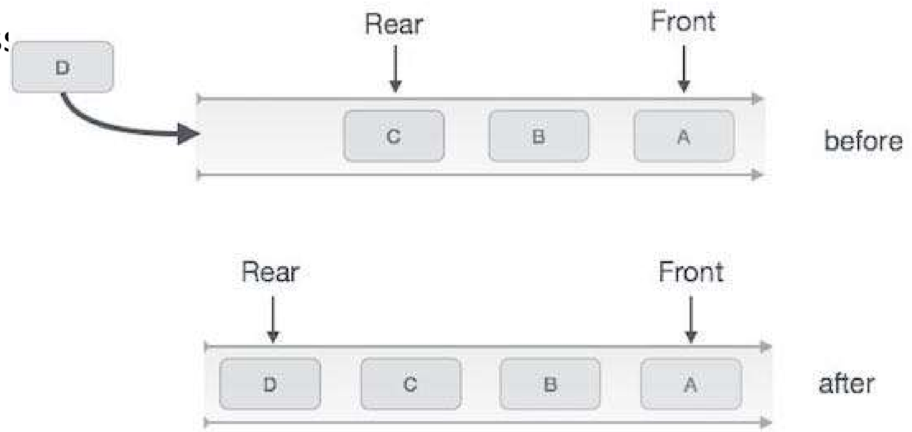
▪ **isEmpty()** – check if stack is empty.

Introduction
Types of DT
Array
Stack
Queue
Linked Lists
Tree and Graph
Hashing
Search & Sort

Insert Operation

(inserting data in the queue at the end /rear position)

- **Step 1** – Check if the queue is full.
- **Step 2** – If the queue is full, produce overflow error and exit.
- **Step 3** – If the queue is not full, increment **rear** pointer to point the next empty space.
- **Step 4** – Add data element to the queue location, where the rear is pointing.
- **Step 5** – return success

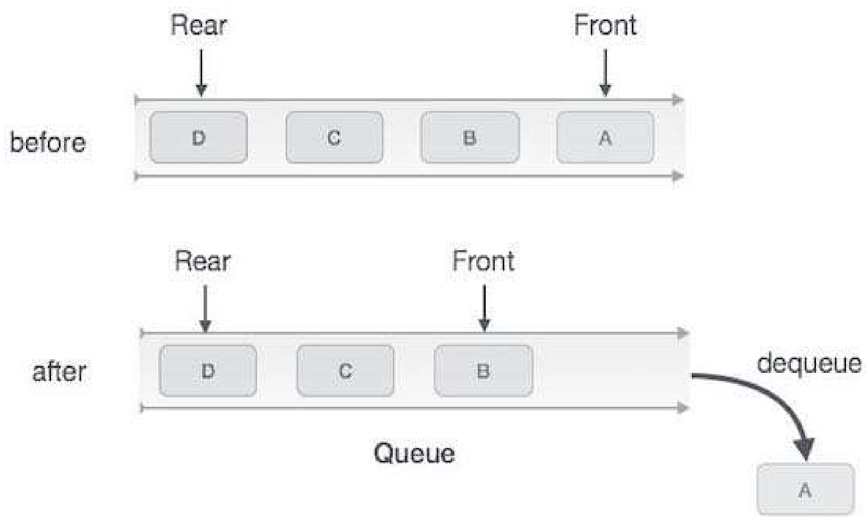


Queue Enqueue

Delete Operation

(Deleting data at the beginning /front position)

- **Step 1** – Check if the queue is empty.
- **Step 2** – If the queue is empty, produce underflow error and exit.
- **Step 3** – If the queue is not empty, access the data where **front** is pointing.
- **Step 4** – Increment **front** pointer to point to the next available data element.
- **Step 5** – Return success.



Queue Dequeue

Variations on Queue

- Priority queue
- Circular queue

Uses of Queue

- Operating systems and Resource management such as CPU scheduling, memory scheduling, printer queue, etc.
- Call centre phone systems
- Scheduling jobs
- Searching