JGI JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

Master of Computer Applications

Advanced Data Structures
(23MCAC101)

Module 2

STACKS & QUEUES

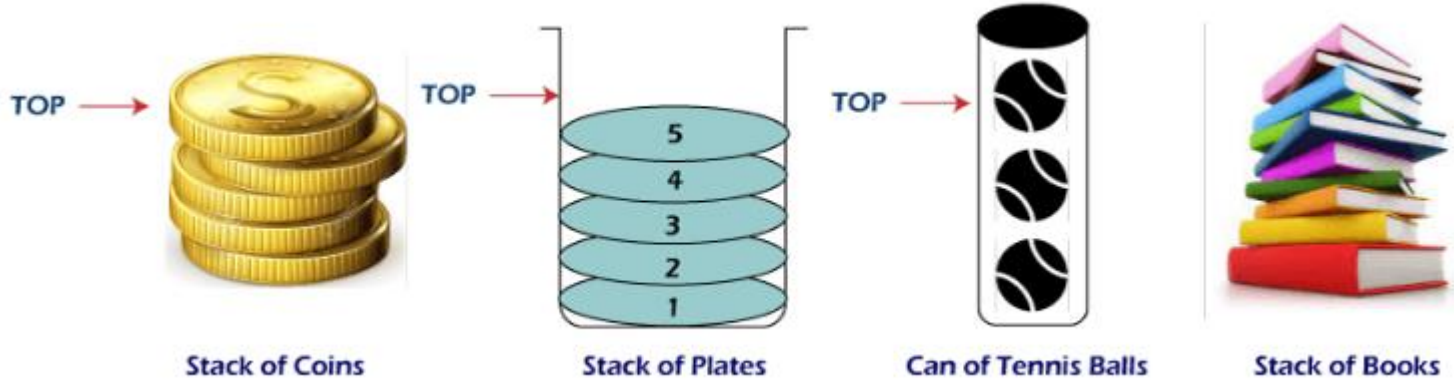
Syllabus – Module2

Stack ADT- Stack operations Push, Pop, Peek, isEmpty, isFull and initialize, Stack implementation using Array, Stack implementation using Linked List, applications of Stacks, evaluating arithmetic expressions, conversion of infix to postfix expression.

Queue ADT – Enqueue, Dequeue, Peek, isEmpty, isFull and initialize, Queue implementation using Array, Queue implementation using Linked List, applications of Queue, Priority Queue — Operations and Applications. Circular Queue & Applications of Queues.

Introduction to Stack

- It is a linear data structure and follows the Last-In First-Out (LIF0).
- Data items are processed in only one end and is known as ‘Top’.
- The real life examples where we can find LIFO style of access



Fundamental operations:

- **Push**:
  - Equivalent to an insert the element on the top of the stack

- **Pop**:
  - Deletes the most recently inserted element
  - Remove the element from the top of the stack

- **Peek**: Examines the most recently inserted element

- **IsEmpty**
  - Check whether the stack is not having any element
- **IsFull**
  - Check whether all cells in the stacks are full with elements
- **Initialize**
  - Initialize the stack is Null to create an empty stack at before the usage

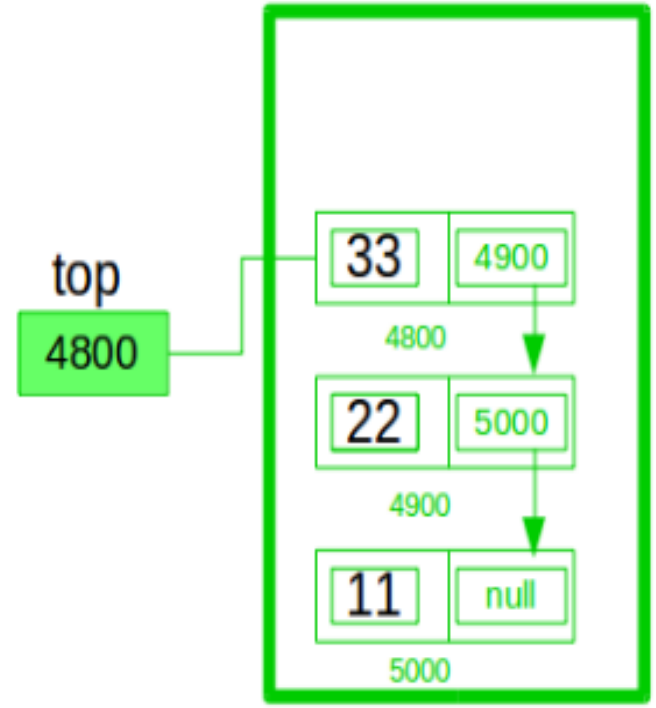
Stack implementation using Array

- Need to declare an array size ahead of time
- Associated with each stack is TopOfStack
  - for an empty stack, set TopOfStack to -1
- Push
  - (1) Increment TopOfStack by 1.
  - (2) Set Stack[TopOfStack] = X
- Pop
  - (1) Set return value to Stack[TopOfStack]
  - (2) Decrement TopOfStack by 1
- These operations are performed in very fast constant time

Stack implementation using Linked List

- Singly Linked List can be used to demonstrate the working principle of Stack
- Push – New node will be inserted in front the list (use 'InsertFront()' module from SLL)
- Pop – 'Head' node will be removed and next node will be a new 'Head' node

  (implement 'DeleteFront()' module in SLL)



Stack – Applications

- Balancing Symbols
- Infix Expression to Postfix Expression
- Postfix Evaluation
- Function calls

Conversion of infix to postfix expression

- Operators +, -, *, /
  - Operators precedence should be considered
- Algorithm

  (1) Make an empty stack.

  (2) Read characters until end of expression

  i. If the character is an operand, send to output

  ii. If it is operator symbol, then push to stack if the top operator precedence is lower than input operator, else pop the operator(s) from the stack until getting lower precedence operator, then push the input operator on top of the stack

  iii. If it is closed symbol, then pop operator and send to output (we pop entries from the stack until we find an entry of lower priority.)

  (3) At end of expression, we pop the stack until it is empty, writing symbols onto the output.

Evaluating arithmetic expressions

- Calculate 4* 1 + 5 + 6 * 1
  - Need to know the precedence rules
- Postfix (reverse Polish) expression
  - 4 1 * 5 + 6 1 * +
- **Use stack to evaluate postfix expressions - Procedure**
  - When a number is seen, it is pushed onto the stack
  - When an operator is seen, the operator is applied to the 2 numbers that are popped from the stack. The result is pushed onto the stack
  - Repeat the above steps until the end of postfix expression
- Example
  - evaluate 6 5 2 3 + 8 * + 3 + *
- The time to evaluate a postfix expression is O(N)
  - processing each element in the input consists of stack operations and thus takes constant time

Queue ADT

Introduction to Queue

Queue ADT

- Like a stack, a *queue* is also a list. However, with a queue, **insertion is done at one end,** while **deletion is performed at the other end**.
- Accessing the elements of queues follows a First In, First Out (FIFO) order.

Example:

- Like **customers standing in a check-out line in a store**, the first customer in is the first customer served.

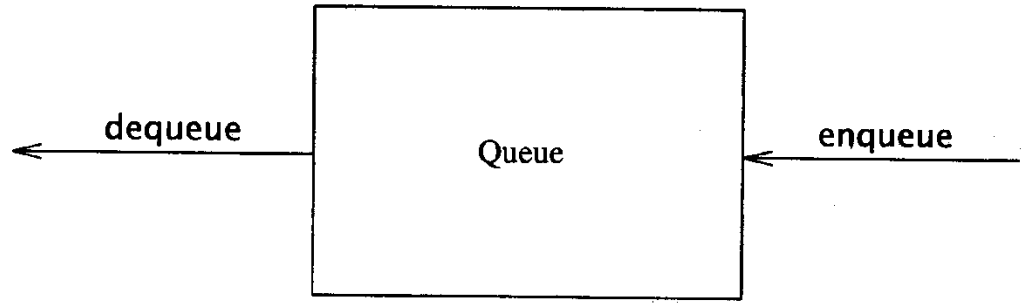
Enqueue and Dequeue operations

- **Enqueue:**

  - insert an element at the **rear end of the list**

- **Dequeue:**

  - delete the element at the **front end of the list**



Peek, IsEmpty, IsFull and Initialize Operations

1. Peek operation

- **Returns the element from the front end** of the queue

2. isEmpty operation

- **Returns true if queue is empty** else returns false

3. isFull operation

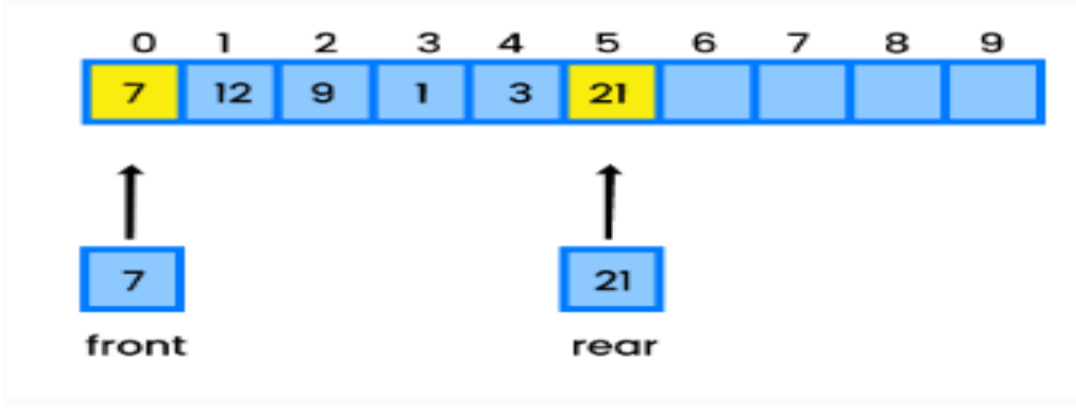
- **Returns true if queue is full** else returns false

4. Initialize operation

- **Assign null to Queue** to **represent the queue is empty** before inserting elements

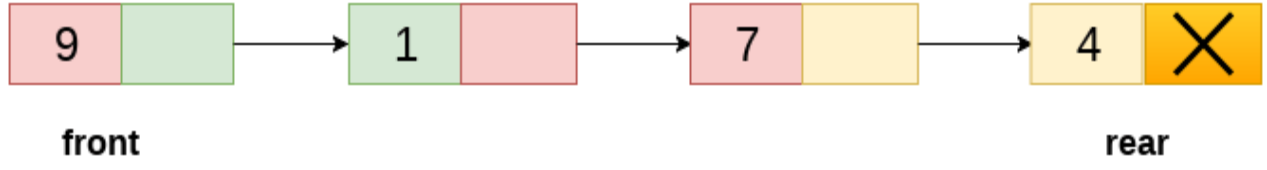
Queue using Array

- **Steps**
  - When enqueuing, the **front index** is always **fixed** and the **rear index moves forward** in the array.
  - When dequeuing, the **element at the front** the queue is **removed.**



Queue using SLL

- Enqueue and Dequeue can be demonstrated using SLL
- **Enqueue** – **Inserting new node in the rear end** of the list module to be used
- **Dequeue** – **Removing Front node** in the list module is to be used



Priority Queue

- a different kind of queue.

Similar to a regular queue:

-**insert in rear**,

-**remove from front**.

-Items in priority queue are **ordered** by **some key**

-Item with the **lowest key / highest key** is always **at the front** from where they are removed.

-Items **then** 'inserted' in **'proper' position**

Idea behind the Priority Queue is simple:

- Is a queue

- But the items are **ordered** by a **key**.

Implies **'position' in the queue may be changed** by the arrival of a new item.

Operations

remove()

So, the **first item has priority and can be retrieved** (removed) **quickly** and returned to calling environment.

Hence, 'remove()' is easy (and will **take O(1) time**)

insert()

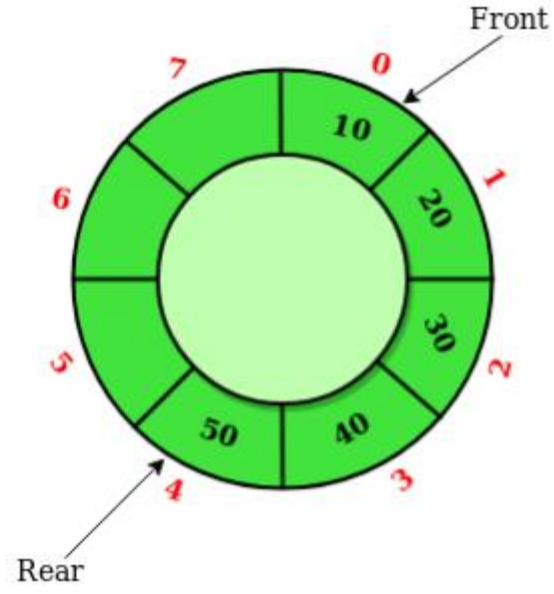
But, we want to **insert** quickly. Must go into proper position.

Implementing data structure: **Array**

- **slow to insert(), but** for
  - small number of items in the pqueue, **and**
  - where insertion speed is not critical,
- this is the **simplest and best** approach.

Circular Queue

- It is an **extended version** **of a** **normal queue** where the **last element of the queue** is **connected to the first element** of the queue forming a circle.
- The operations are performed based on **FIFO (First In First Out)** principle. It is also called **'Ring Buffer'**.



Operations on Circular Queue

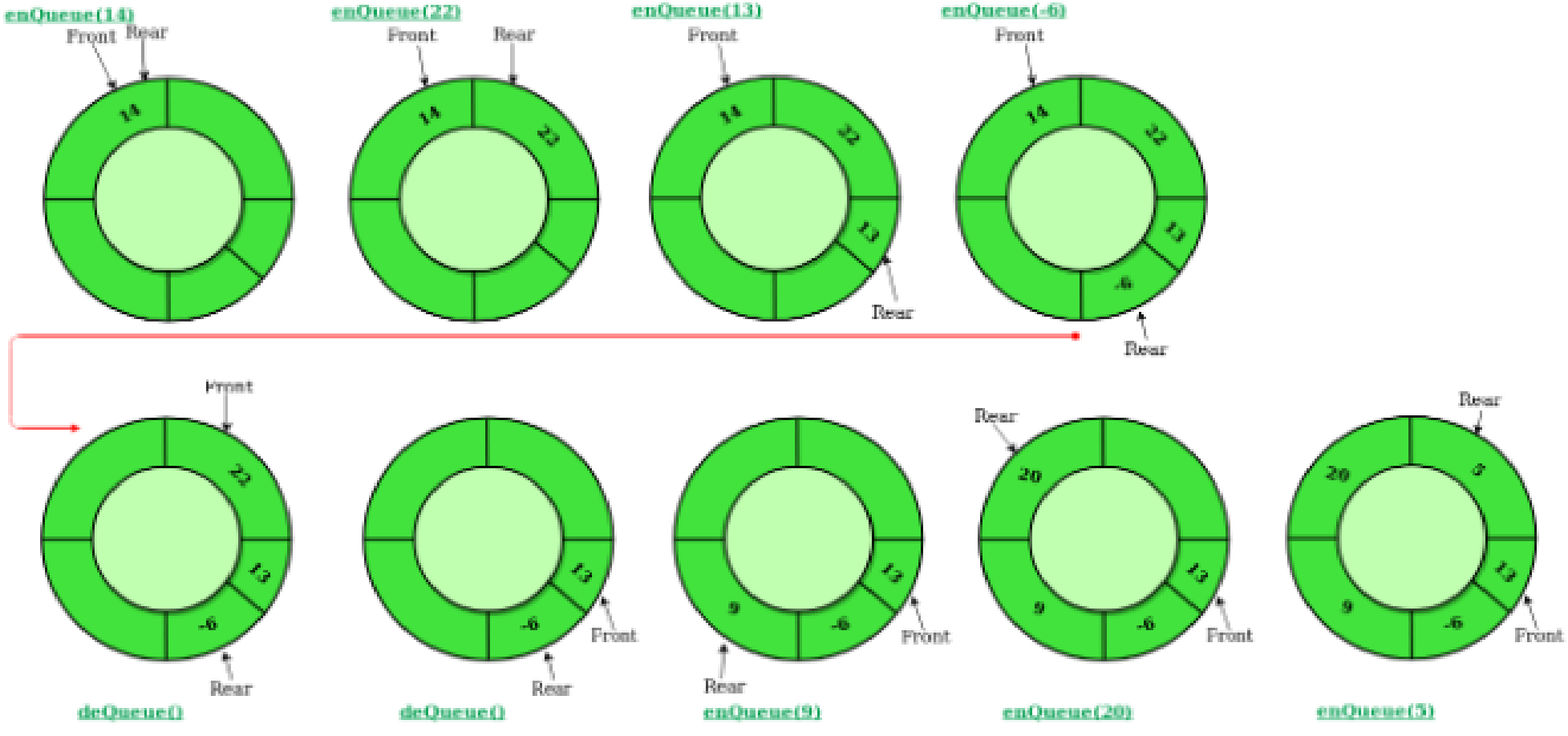
- **enQueue(value):** To **insert an element** into the circular queue.

  In a circular queue, the **new element** is always **inserted at the rear position**.

- **deQueue():** To **delete an element** from the circular queue.

  In a circular queue, the element is always **deleted from the front position**.

Introduction to DS



Working of Circular queue operations

Applications of Queue ADT

- Used for **process scheduling in OS**
- Used in **printer devices**
- Used in **Network Protocols** operations
- Used to implement **BFS traversal technique**
- Used to implement **Customer care application**
- Implementing '**Ticket Reservation System**'.
- And etc..