Stacks & Queues

- **A linear data structure** represented by a sequential collection of elements in a **fixed order**
- **Dynamic size.**
- Contain elements of **different data types.**
- Random access of elements are not allowed
- Implement using list or linked list

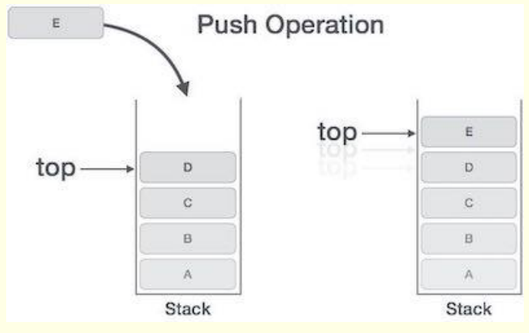
Stack	**Queue**
LIFO – Last In, First Out	**FIFO** – First In, First Out
Only the **top** element can be accessed	Only the **front** element can be accessed.
It has only one pointer- the **stack pointer** This pointer indicates the address of the topmost element or the last inserted one of the stack.	Two pointers – **front** and **rear**
push(data) − Pushing (storing) an element on the stack. **pop()** − Removing (accessing) an element from the stack. **peek()** − get the top data element of the stack, without removing it. **isFull()** − check if stack is full. **isEmpty()** − check if stack is empty	**enqueue(data)** – Adds to the rear **dequeue()** – Removes from the front **isFull()** – check if queue is full **isEmpty()** – check if stack is empty
Browser history (Back button) Undo operations in text editors Function call stack (program execution)	Printer queue (print jobs in order) Customer service queue Task scheduling (e.g. CPU process queue)

Stack Algorithms

When the Stack pointer is pointing to the top element in the stack

Stack PUSH Algorithm

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 to next empty space.
Step 4 − Adds data element to the stack location, where top is pointing.



#If the stack is full an error message will be generated:

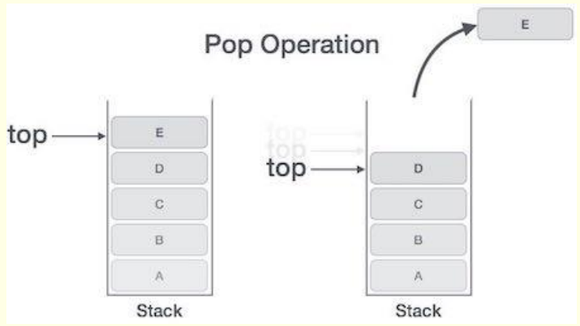
```
if isFull() then
      print 'stack overflow'
else
      # else add 1 to the stack top pointer
      top=top+1
      #insert new item to the top of the stack
      stack[top] = item
end if
```

isFull() function

You could use:
if stack.length == MAX:

Stack Pop Algorithm

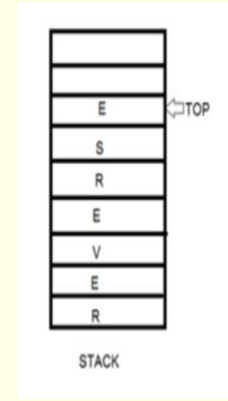
Step 1 − Checks if the stack is empty.
Step 2 − If the stack is empty, produces an error and exit.
Step 3 − If the stack is not empty, access the data at which **top** is pointing.
Step 4 − Decreases the value of top by 1
Step 5 – Return item



#If the stack is empty an error message will be generated
```
if isEmpty() then
      print 'stack underflow'
else
      # else get the item to pop
      item = stack[top]
       #change the top of the stack
      top=top-1
      return item
end if
```

isEmpty() function
You could use:
if top == -1
if stack.length == 0

Display items in Stack algorithm




```
if isEmpty() then
      print 'stack empty'
else
      #only loop though the list from
          # the top to 0
      for i=top to 0
         print s[i]
      next i
end if
```

Notes for Exam:

Here is an example of an exam question where you only need to use the procedures push() or pop()

A function, push, can be used to add a character to a stack. For example:

theStack.push("H")

places the character H onto the stack, theStack.

A procedure, pushToStack, takes a string as a parameter and pushes each character of the message onto the stack, messageStack.

Complete the procedure below.
Solution:

```
procedure pushToStack(message)
   for x = 0 to message.length() //loop through each
                                 //letter
      messageStack.push(message.substring(x,1)) //take
            //each character and push onto stack
   next x  //move to next letter
endprocedure
```

Alternative Representation of a Stack

In the exam, the **stack pointer** may be pointing to the **next available free space.**

top	3

index	stackItem
9	
8	
7	
6	
5	
4	
3	
2	33
1	14
0	20

```
Push()
if stackPointer > 9 then
      print 'stack overflow'
else
      #insert new item to the top of the stack
      stack[top] = item
      # add 1 to the stack top pointer
      top=top+1

end if
```

```
Pop()
If stackPointer == 0 then
        print 'stack underflow'
else
        #change the top of the stack
        top=top-1
       # get the item to pop
        item = stack[top]
        return item
end if
```

Notes for exam:

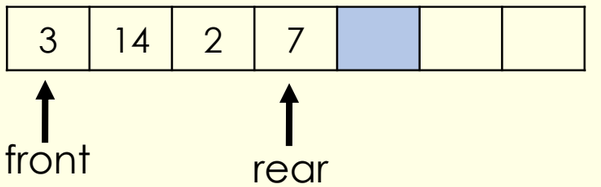
1. Is the top pointer pointing to the item at the top of the stack or the next empty space?
2. Use the identifiers provided
3. Think logically about determining the isEmpty and isFull condition

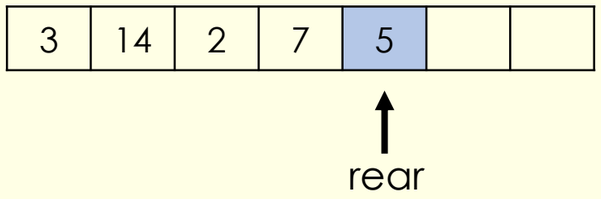
Queue Algorithms

Enqueue Algorithm in a linear queue (enqueue)

- enqueue(item)will put the given data at the rear of the current queue.

1. Check if queue is full
2. If full output error and stop
3. Else increment rear pointer
4. Insert new data item into the rear pointer position.





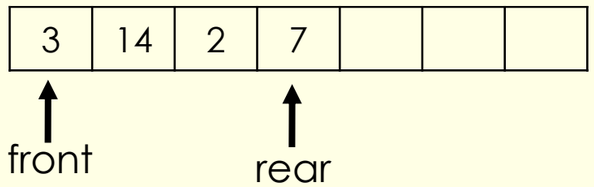

```
if isFull() then
     print ("overflow") # error message if queue is full

else
     rear = rear + 1
     queue[rear] = data
end if
```

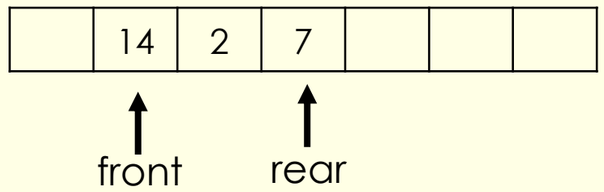
Working out the isFull() function
It can be done in a number of different ways, for example:
If rear == maxsize – 1
OR
By using a counter size to keep a track of data items
if size == MAX

Dequeue algorithm in a linear queue

- dequeue()will remove the item at the front of the queue and return it for use in the main program.

1. Check if queue is empty
2. If empty output error and stop
3. Else copy data from the front pointer position
4. Increment front pointer
5. Return data






```
if isEmpty () then
    print ("queue is empty")
else
    data = queue[front]
    front = front + 1
    return data
```

Working out the isEmpty() function
It can be done in a number of different ways, for example:
if front > rear
  front = 0
  rear = -1
Or by using the size counter :
if size = 0

Displaying a linear queue

```
#only loop though the list from the front to the rear pointer
        for i=front to rear
            print q[i]
        next i
```

Notes for exam

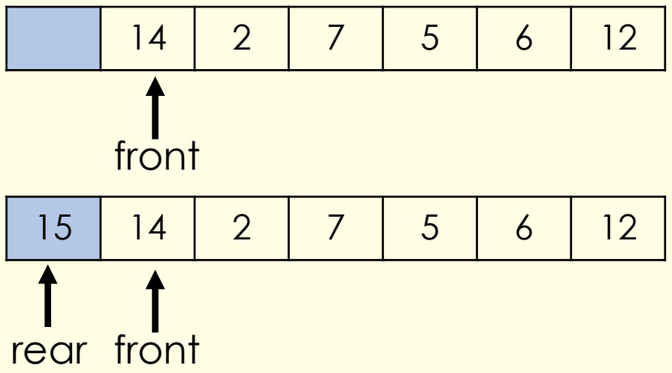
- Front and rear pointers can be either way round …
- Pointers might be referred to as head/tail or front/rear or front/rear
- It can either be a circular and linear queue – it will specify this in the exam question

Circular Queue

This means data is added at the rear of the queue can be stored in locations vacated at the front of the queue.

Enqueue Algorithm in a circular queue

1. Check if queue is full
2. If full output error and stop
3. Else if rear is equal to maxsize set rear pointer to 0
4. Else increment rear pointer by 1
5. Insert new data item into the rear pointer position.
6. Increment size by 1



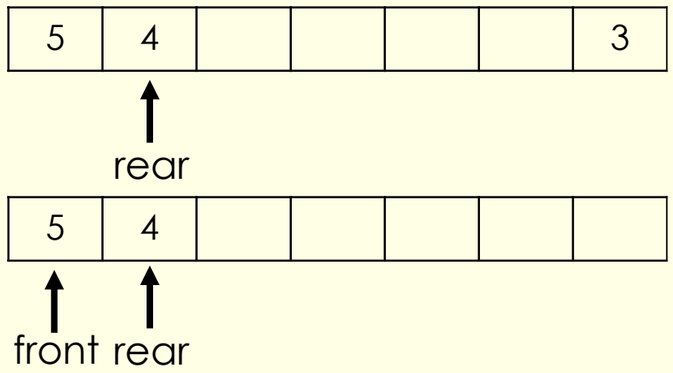

```
                                                    # error
message if isFull is True
If isFull() then
    print overflow
else
      if rear = maxsize - 1
         rear= 0
      else
         rear = rear + 1
      queue[rear] = data
      size =size + 1
end if
```

isFull() function may look something like this. The program uses a variable, i.e. size that keeps track of adding and removing items from the queue, so:

if size == maxsize:
    return True

Dequeue Algorithm in a circular queue

1. Check if queue is empty
2. If empty output error and stop (reset front and rear pointers)
3. Else copy data from the front pointer position
4. If front pointer is equal to maxsize then resent front pointer to 0
5. Else increment front pointer by 1
6. Decrement size by 1
7. return data




```
# error message if queue is empty
if isEmpty() then
   print empty
   front = 0
   rear = -1
else
   data = queue[front]
   if front = MAXSIZE -1 then
        front = 0
   else
        front = front +1
   endif
   size = size - 1
   return data
```

Working out the isEmpty() function can be done in a number of ways, for example, the program uses a variable, i.e. size that keeps track of adding and removing items from the queue, so:

if size = 0

Displaying a circular queue

Displaying a circular queue is just a little bit more complicated.

It has to print from the front to the rear taking into account that more items may have been added to the front of the list

```
If isEmpty() then
   print("Queue is empty")
else
   temp = front
   for i = 0 to size - 1
      print q[temp]
      temp = temp + 1
      if temp == maxsize then
         temp = 0  // wrap around manually
      end if
   end for
end if
```

Notes for exam:

- Front and rear pointers can be either way round …
- Pointers might be referred to as head/tail or front/rear or front/rear
- It can either be a circular and linear queue – it will specify this in the exam question

Priority queue

A priority queue is a type of abstract data structure where each element has a priority. Instead of being processed in the order they were added (like a regular queue), elements are processed based on their priority.

Key Features:

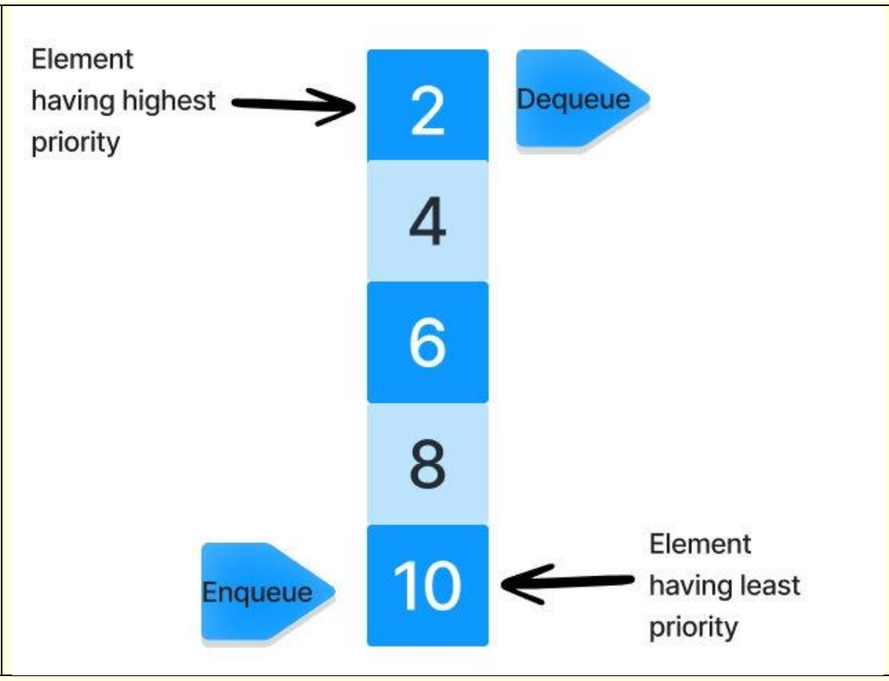
- Each item in the queue has:
- A value
- A priority level
- The item with the highest priority is removed first.
- If two items have the same priority, they are processed in FIFO (First In, First Out) order.

Operations:

- Insert (enqueue): Add an element with a priority
- Remove (dequeue): Remove the element with the highest priority
- May be implemented using:
- Arrays/lists (less efficient)
- Heaps (more efficient, e.g., binary heap)

Example Use Cases:

- Task scheduling (e.g., CPU processes)
- Dijkstra's algorithm (for shortest path in graphs)
- Emergency room triage systems



Example of implementing a stack structure

```
# Procedural Linear Stack Implementation
# Initialize
maxsize = 5
stack = [None, None, None, None, None]
top = -1

procedure push(stack, data, top, maxsize)
    if isfull(top, maxsize) then
        print("Error: stack is full")
    else
        top = top + 1
        stack[top] = data
    endif
    return stack, top
endprocedure

function pop(stack, top)
    if isempty(top) then
        print("Error: stack is empty")
        return None, top
    else
        item = stack[top]
        top = top - 1
        return item, top
    endif
endfunction

procedure displaystack(stack, top)
    if isempty(top) then
        print("Error: stack is empty")
    else
        print("stack contents:")
        for i = 0 to top
            print(stack[i])
        next i
    endif
endprocedure

function isfull(top, maxsize)
    if top = maxsize - 1 then
        return True
    else
        return False
    endif
endfunction

function isempty(top)
    if top = -1 then
        return True
    else
        return False
    endif
endfunction

function peek(stack, top)
    if isempty(top) then
        return "Error: stack is empty"
    else
        return stack[top]
    endif
endfunction

# Main program
stack, top = push(stack, "cat", top, maxsize)
displaystack(stack, top)
item, top = pop(stack, top)
print("popped item: " + item)
```

```
# OOP Linear Stack implementation

class stack
    # constructor
    public procedure new()
        # initializing stack
        maxsize = 5
        s = [none, none, none, none, none]
        top = -1
    endprocedure

    procedure push(data)
        if isfull() then
            print("error: stack is full")
        else
            top = top + 1
            s[top] = data
        endif
    endprocedure

    function pop()
        if isempty() then
            print("error: stack is empty")
        else
            item = s[top]
            top = top - 1
            return item
        endif
    endfunction

    procedure displaystack()
        if isempty() then
            print("error: stack is empty")
        else
            print("stack contents:")
            for i = 0 to top
                print(s[i])
            next i
        endif
    endprocedure

    function isfull()
        if top = maxsize - 1 then
            return True
        else
            return False
        endif
    endfunction

    function isempty()
        if top = -1 then
            return True
        else
            return False
        endif
    endfunction

    function peek()
        if isempty() then
            return "error: stack is empty"
        else
            return s[top]
        endif
    endfunction
endclass

# main program
s = new stack()
s.push("cat")
s.displaystack()
print("popped item: " + s.pop())
```

Example of implementing a queue data structure

```
# Procedural Linear Queue Implementation
# Initialize an empty queue with a fixed size
maxsize = 5
q = [None, None, None, None, None]
front= 0
rear=-1

# Enqueue operation
function enqueue(q, item, rear, maxsize)
  if isFull(rear,maxsize):
    print("Queue is Full")
  else:
    rear = rear + 1
    q[rear] = item
    return q,rear
  endif

function isFull(rear,maxsize):
  #Checks if the queue is full
  if rear == maxsize-1
    return True
  else
    return False
  endif

# Dequeue operation
function dequeue(q, front, rear)
  if isEmpty(front,rear)
     front = 0
     rear = -1
     print ("Queue is empty")
  else
     data = q[front]
     front = front + 1
   endif
  return q, front, rear, data

# Check if the queue is empty
function isEmpty(front, rear)
  if front > rear
    return True
  else
    return False
  endif

# Peek operation
function peek(q, front)
   return q[front]

function displayQueue(q, front, rear)
    for i=front to rear
        print q[i]
    next i

#Main program
q,rear = enqueue(q, "cat", rear, maxsize)
print("Display.....")
displayQueue(q, front, rear)
q, front, rear, data = dequeue(q, front, rear)
displayQueue(q, front, rear)
q, front, rear, data = dequeue(q, front, rear)
```

```
# OOP Linear Queue Implementation
class Queue:

    #Constructor
    public procedure new()

        # initializing queue with none
        maxsize = 5
        q = [None, None, None, None, None]
        front = 0
        rear = -1

    public procedure enqueue(data)
        if isFull(). #Check whether queue isFULL. (rear == maxsize-1)
            print("Queue is Full")
        else
            rear = rear + 1#then increment rear value by one
            q[rear] = data #set queue[rear] = data
         endif

    public function isFull()
        #Checks if the queue is full
        if self.rear == self.maxsize-1
            return True
        else
            return False
        endif

    public function dequeue()
        #Check whether queue is EMPTY
        If isEmpty()
            print ("Queue is empty")
        else
            data = q[front] #get data using front pointer
            front = front + 1 # increment the front value by one.
            return data
        endif

    public function isEmpty()
        #Checks if the queue is empty
        if  front == rear
          front = 0 #reset front and rear pointers
          rear = -1
          return True
        else
          return False

    public function peek()
         ##peek() method will return the first item in the queue
        return q[front]

    public procedure displayQueue()
        if isEmpty()
            print("Queue is empty")
        else
            for i=front to rear
                  print q[i]
            next i
        endif


#Main program
  #create a new instance of a queue object
    q = Queue()
    q.enqueue("cat")
    print("Display.....")
    q.displayQueue()
    q.dequeue()
```