DS

Circular Linked Lists
Circular Arrays
Stacks and Queues

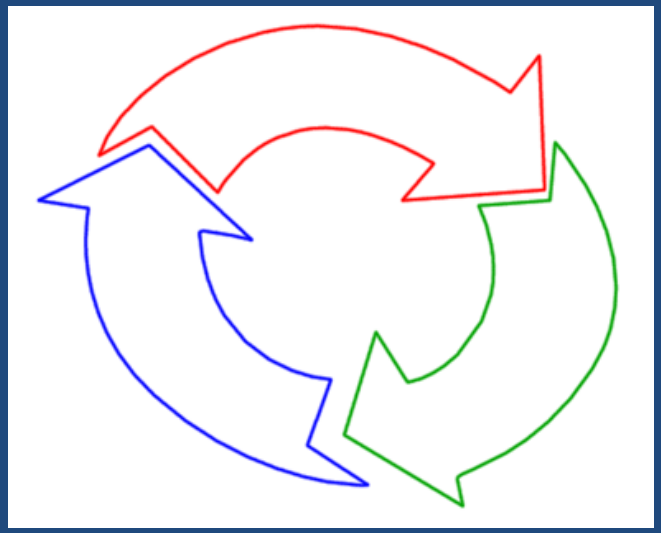
Circular Linked Lists

Circular Linked Lists

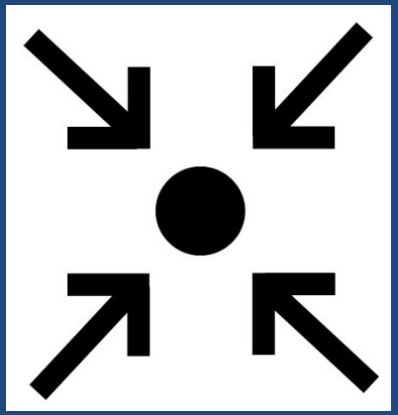
The list

never

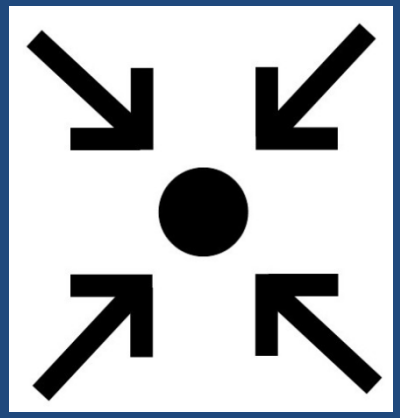
ends



The final *next* pointer
points to
the first item in the list



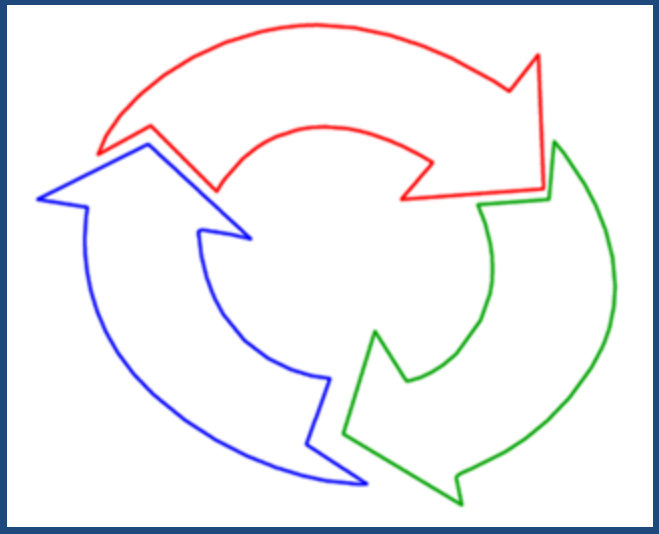
If the list is doubly-linked, the first item's *prev* pointer points to the last item



There is no *tail* pointer

Traversal

Traversal loops back to the start



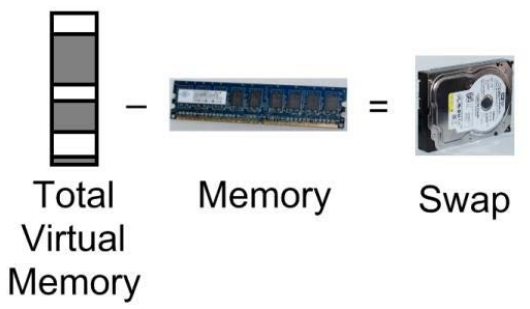
Compare
the next pointer
of the current node
to
the head pointer

You can't just compare the current node to the head pointer or you'll never start!

Example of Usefulness of a Circular List

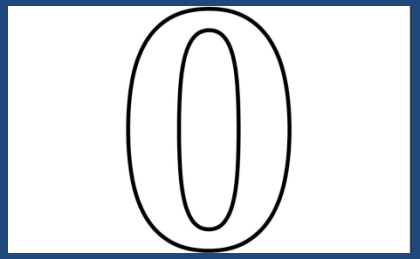
Determine which memory page is least recently used, so that it can be swapped out

(You took about this in OSF)



Uses *reference number*

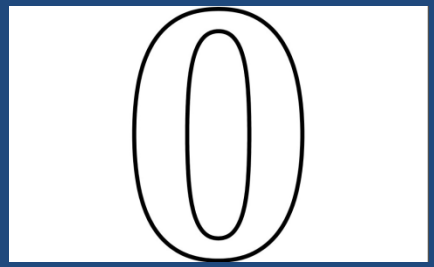
Set to 1 when the page is accessed and 0 otherwise



(Internal to the memory manager)

Commentary: Mechanism of Swapping

When swapping out a page, the memory manager searches for a page with a *reference number* of 0



While it's searching, it resets the *reference number* of the pages it found from 1 to 0

Commentary: Why Circular?

Will always find
a page to swap out
because it'll either find
one with a value of 0
or
it will find where it started
(which was reset from 1 to 0)

Commentary: Optimization of the Search

It'll then reset the head of the list to point to the location we've just swapped out

The actual location
of the start of the list
isn't important
since we're just using it
to try to find a swappable page

Search from here next time

because

we didn't necessarily

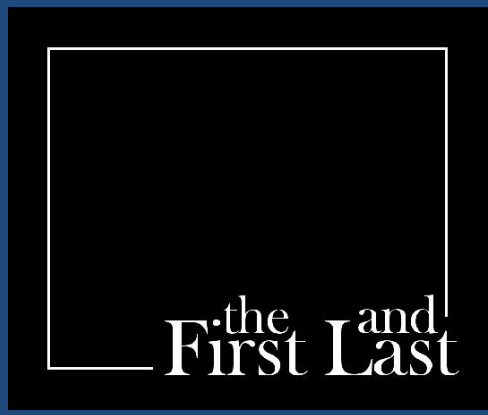
complete the circuit this time

Circular Arrays

Circular Arrays

Arrays in which
the "first" element in the array
is not necessarily
the physically-first one
and the "last" element
is not necessarily
the physically-last one

Have
a head and tail **index**
(not pointer)
that indicate where
the “first”
and “last” elements
actually are



When you reach
the physical end of the array
in traversal,
loop around
to the physical start
of the array

Circular Array Terminology

The "physical-first" element of a circular array called *fred* would be *fred[0]*

The "logical-first" element of a circular array is the first element that would be processed

Similarly:

- The "physical-last" element of a circular array of size 10 called *fred* would be *fred[9]*
- The "logical-last" element of a circular array is the last element that would be processed

More Circular Array Terminology

The "head" of a circular array is the index of the "logical-first" element

The "tail" of a circular array is the index of one **past** the "logical-last" element

You could do these with pointers if you want, as well

Circular Array: Full vs. Empty (one definition)

A circular array is full if the tail is one position before the head

This must account for wrapping around the physical end of the array

A circular array is empty if the head and tail are equal

This implies that we cannot use up all of the elements in the array

This is one method of implementation

You also could keep a separate variable indicating whether the list is full, empty, or neither

Examples of Full and Empty in this Circular Array

Let's say we have an array of size 5 (indeces 0 to 4)

"X" will indicate a used element

Index	0	1	2	3	4
Value					

Example #1

Index	0	1	2	3	4
Value	X	X	X		

Head = 0

Tail = 3

This is not empty and not full

Example #2

Index	0	1	2	3	4
Value	X	X	X	X	

head = 0

tail = 4

This is full

Example #3

Index	0	1	2	3	4
Value	X		X	X	X

head = 2

tail = 1

This is full

Example #4

Index	0	1	2	3	4
Value					

head = 3 (or any valid index)

tail = 3

This is empty

Example #5

Index	0	1	2	3	4
Value		X	X	X	X

head = 1

tail = 0

This is full

Stacks and Queues

Reading

A Book on C, Sections 10.5 to 10.7

Why?

Queues in real life:

- e.g. lineups while shopping or filling the car with gas
- e.g. sending an application in

Stacks in real life are less obvious

- e.g. an "IN box" that stuff piles up in
- e.g. a narrow drawer that you throw stuff into

Sorting

These are sorted lists in which the order of the list is time-based

The sorting is implicit,

in that

it happens through

the act of adding to the list

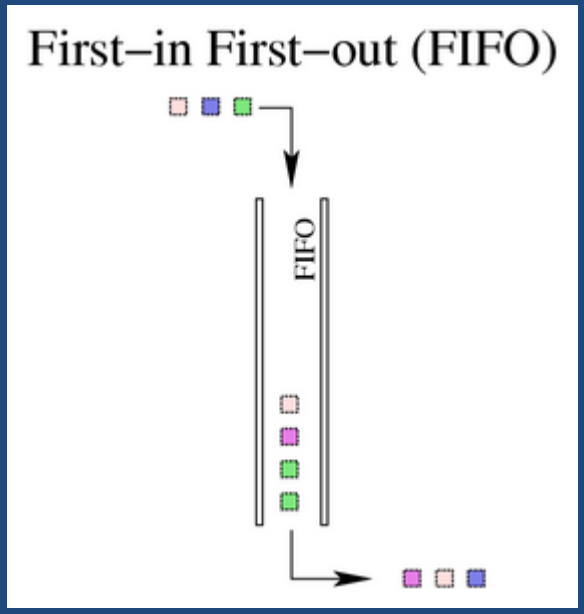
There is the potential of delayed processing of the items

So there may be more than one item in the list

FIFO vs. LIFO

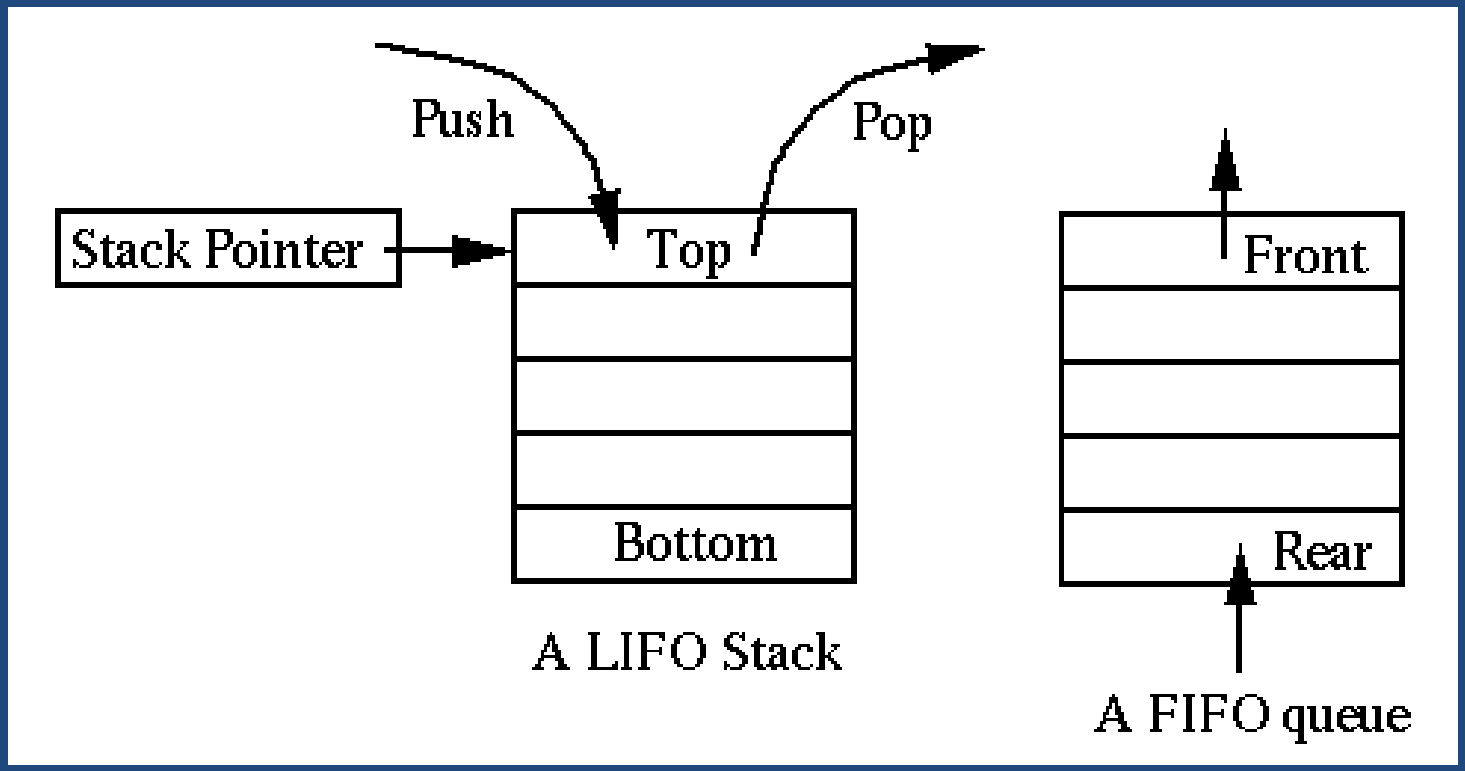
In a queue, the **first** item into the queue is the first item processed from the queue

- This is called "FIFO" for "First In, First Out"


First–in First–out (FIFO)
FIFO

In a stack, the **last** item into the stack is the first item processed from the stack

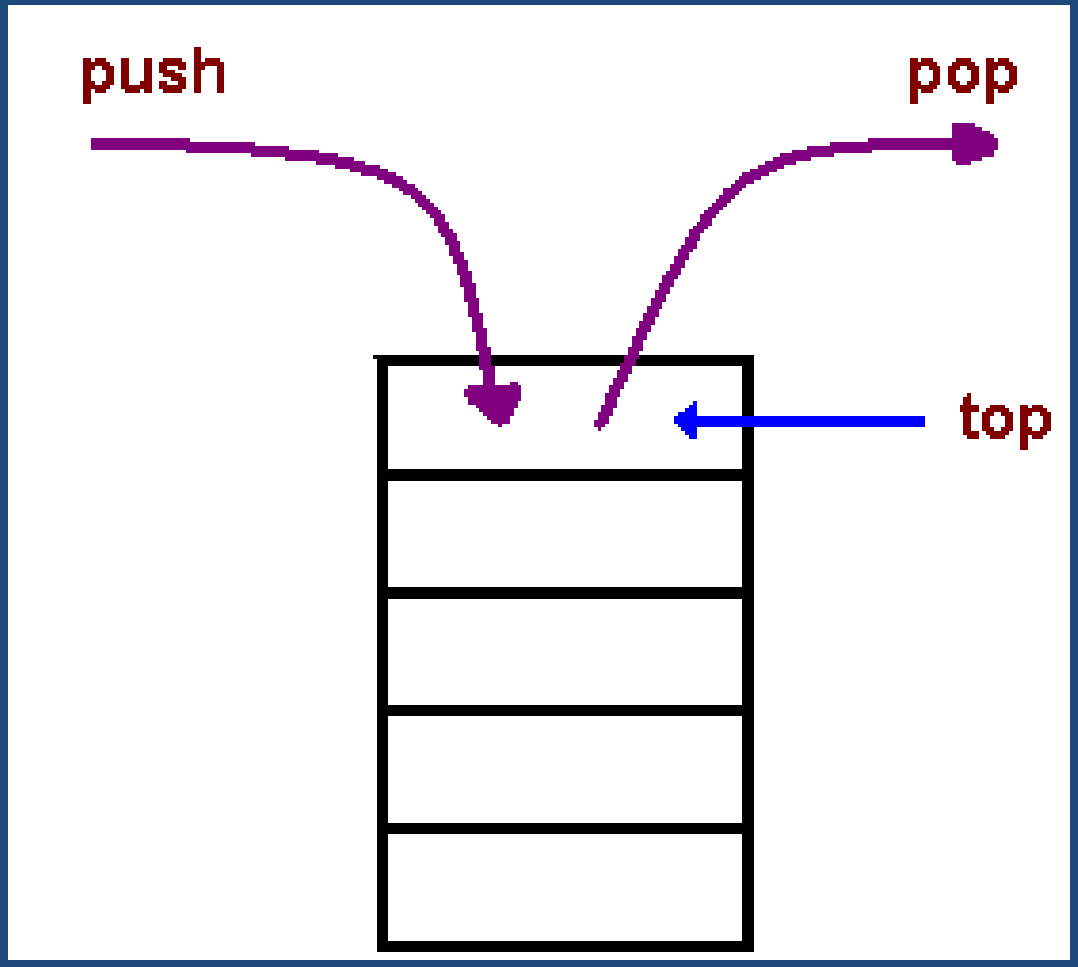
- This is called "LIFO" for "Last In, First Out"


Push
Pop
Stack Pointer
Top
Bottom
A LIFO Stack
Front
Rear
A FIFO queue

Stack Terminology

Putting an item on the top of the stack is called "pushing" it

Taking an item off the top of the stack is called "popping" it


push
pop
top

Viewing an item on the stack is called "peeking" at it

Queue Terminology

If you put at item at the end of a queue, you "enqueue" it

If you take an item from the start of a queue, you "dequeue" it

Looking at an item in the queue without processing it is called "peeking" at it

(same as stack)

Implementation

So far, stacks and queues have been referred to as "lists" but **not** "linked lists"

You **can** use a linked list to implement them but you don't **have** to

Other alternatives if imposing a maximum size is OK:

arrays

circular arrays (queues only)

Actions in a Queue

Queue items are always
added to the tail
and
processed from the head

Processing a queue element always involves deleting it

Typically, there is no need for:

- deletion of items except at the head
- adding of items except at the tail
- explicit sorting

Actions in a Stack

Actions are restricted in a stack, similar to in a queue

Difference: Items are always added to the tail and processed from the **tail**

So, you only have to support deletion from the **tail**

Stack Overflow

If using an array to implement a stack, you have to prevent stack overflow

Don’t add past the maximum size of the array

Summary

Circular Linked Lists

Circular Arrays

Stacks

Queues

all have their uses