# Collections

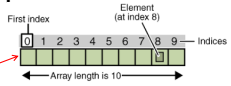
## Collections/Containers/Data Structures

- Containers for like objects – building blocks for most programs
- We will focus on the following commonly used structures
  - Array
  - List
  - Stack
  - Queue
  - Dictionary
  - Linked List

## Array – Characteristics

- collection of items of same type
- in contiguous memory locations
- random access
- memory address calculations, given an index, are built into the CPU electronics – fairly primitive structure
- fixed size

## Array – Memory Representation




`int[] a;`
- `a` is allocated on the stack
- value of `a` is `null`

`int[] a = new int[10];`
- `a` is allocated on the stack
- value of `a` is base address of structure created in the heap

`a[8]`
- address = base address + size of type * index

## Array – Common Operations

- Allocation

```
int[] a = new int[4]
int[] a = {1, 2, 3, 4}
```

- Access

```
a[index] // where 0 <= index < a.Length
```

- Traversal
  - for loop
  - foreach loop

## Array – Performance and Usage

- read and write using index – fast, random access
- limits: fixed size and homogeneity
- use when: number of elements (size) is predetermined (calculable) and direct access is required
- alternative: ArrayList class
  - not a fixed size
  - performance not as fast

## List – Model

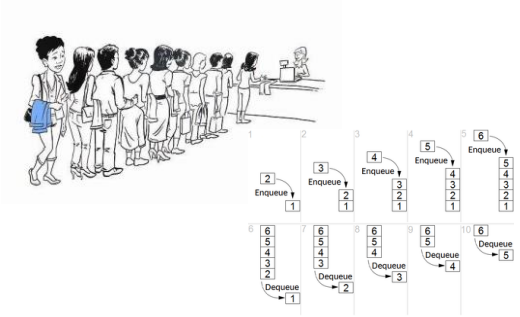

## List – Characteristics

- random access
- no fixed capacity – dynamically resizes
- use indexing to access or set values
- generic version of ArrayList

## List – Common Operations

- Allocation
- Access
- Traversal

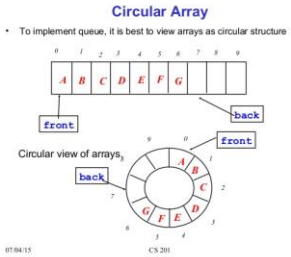
## Queue – Model




## Queue – Characteristics

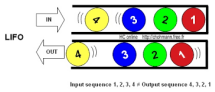
- FIFO (first in, first out) processing, i.e. no random access
- no fixed capacity – dynamically resizes
- heterogeneous structure

## Queue – Memory Representation



## Queue – Common Operations

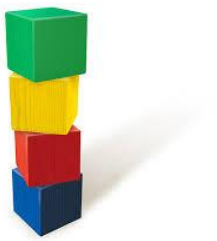
- Enqueue – add item at the end of the line (back, tail)
- Dequeue – remove item from the front of the line
- Peek – look at the item at the front of the line
- Contains – is an item in the queue?




## Queue – Performance and Usage

- Used for FIFO processing. Examples: print spoolers, messaging, job schedulers
- Class handles changes in capacity automatically

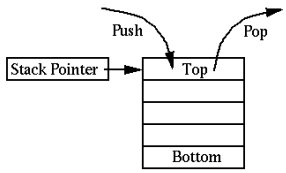
## Stack – Model



## Stack – Characteristics

- LIFO (last in, first out) processing, i.e. no random access
- no fixed capacity – dynamically resizes
- heterogeneous structure

## Stack – Memory Representation



A LIFO Stack

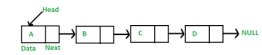
## Stack – Common Operations

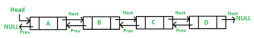
- Push – add item to top of stack
- Pop – remove item from top of stack
- Peek – look at the item on the top of the stack
- Contains – is an item in the stack?

## Stack – Performance and Usage

- Used for LIFO processing. Example: method call tracking, “undo list”
- Class handles changes in capacity automatically

## Linked List – Model



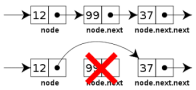

## Linked List – Characteristics

- A collection of nodes where each node has a pointer or reference to the next node in the list.
- Nodes do not occupy contiguous locations
- No size constraints or needs for memory reallocation as grows

## Linked List – Common Operations

- Add


- Remove

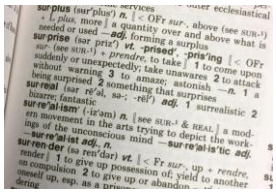



- Find

## Linked List – Performance and Usage

- No random access
- Slow at retrieving data because have to walk the list to find item
- Fast insertion and deletion
- No capacity limit
- Maintain order as insert or delete
- Example: priority queue

## Dictionary – Model



## Dictionary – Characteristics

- collection of key, value pairs
- keys are unique
- also called “associative array”, “map”, “symbol table”
- random access via “key”
- values not in contiguous memory locations
- no fixed capacity
- heterogeneous structure

## Dictionary – Common Operations

- Add pair
- Remove pair
- Modify existing value
- Lookup of value using key
- ContainsKey
- ContainsValue