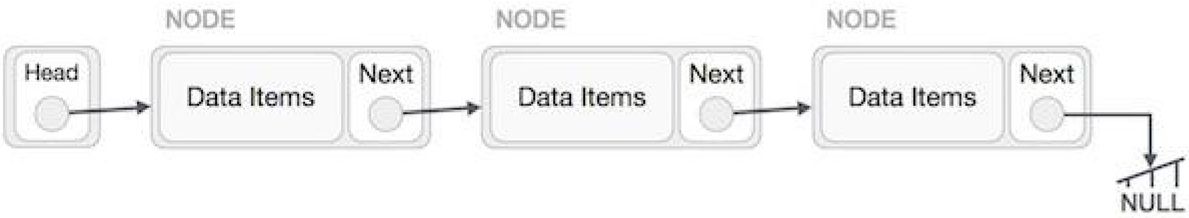
Data Structures

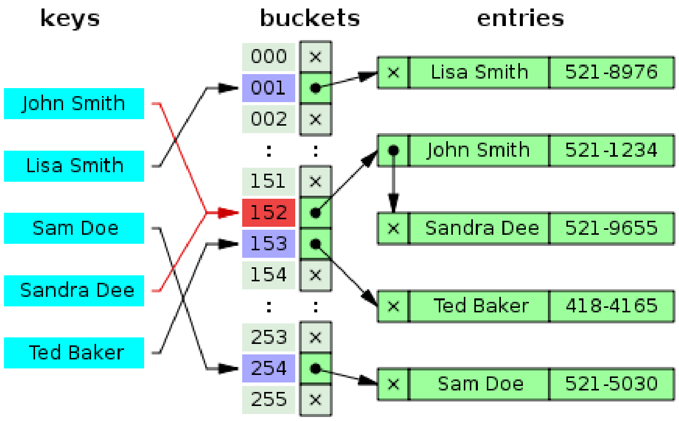
- Linked list
    - Linear data structure in which the elements are not stored at contiguous memory locations
        - Connected using pointers
        - Consists of nodes where each node contains a data field and a reference to the next node



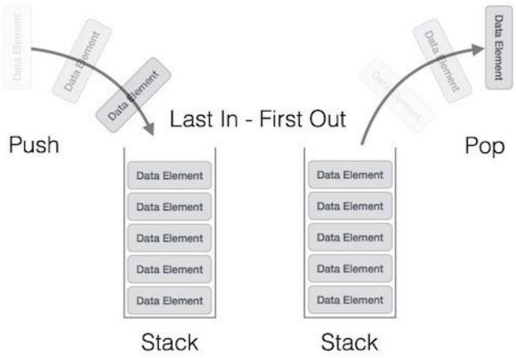
-
    - Basic operations that are supported:
        - Insertion - adds an element at the beginning of the list
        - Deletion - deletes an element at the beginning of the list
        - Display - displays the complete list
        - Search - searches an element using the given key
        - Delete - deletes and element using the given key
- Dictionary/Map
    - General-purpose data structure for storing a group of objects
    - Has a set of keys and each key has a single associated value
    - The keys must be simple types, while the values can be any type
    - Keys MUST be unique
        - A duplicate key
    - Items in a dictionary are arbitrary– any sort of loop will return arbitary values
    - *Abstract Data Type-*
        - A type or class for objects whose behavior is defined by a set of values and a set of operations
            - Only behavior is defined, not implementation
- Set
    - Where a dictionary is a structure of key-value pairs, a set stores various elements in a row
        - Stores any number of unique values (of the same type) in any order you wish
        - Differ from arrays because they only allow non-repeated, unique values

```
>>> an_array = [1,2,2,3,3,4] # repeated values
>>> a_set = set(an_array) # non-repeated, unique values
>>> a_set
{1, 2, 3, 4}
```

- Hash table
    - Loosely type (non-generic) collection
        - It stores key-value pairs of any data types
    - A hash table is a possible implementation of a dictionary



- Stack
    - Considered an ADT
    - Behaves like a real world stack
        - Deck of cards
        - Stack of plates
    - Allows operation at one end only, at a given time we can only access the top element of the stack
    - LIFO data structure
        - Last-in-first-out
            - The element which is placed last is access first
            - Insertion is called PUSH
            - Removal operation is called POP



- Queue
    - ADT similar to stacks
        - Unlike stacks, a queue is open at both ends
    - Enqueue
        - The end that is always used to insert data
    - Dequeue
        - The end that is used to remove data
    - Follows first in first out methodology, the data item stored first will be accessed first
- Priority queue
    - A type of queue in which each element is associated with a priority value
        - Elements are served on the basis of their priority
    - Typically the element with the highest value is considered the highest priority element
        - The element with the highest priority is removed first instead of the typically last in first out rule for classical queues
- Tree
    - Represents nodes connected by edges
        - A node is a basic unit of a data structure
    - Important terminology to a tree
        - Path
            - Refers to the sequence of nodes along the edges of a tree
        - Root
            - The node at the top of the tree. There is only one root per tree and one path fom the root node to any other node
        - Parent
            - Any node except the root node has one edge upward to a node
        - Child
            - The node below a given node connected by its edge downward
        - Leaf
            - The node which does not have any child node
        - Visiting
            - Refers to checking the value of a node when control is on the node
        - Traversing
            - Means passing through nodes in a specific order
        - Levels
            - Respresnets the generation of a node. If the root node is at level 0, the its next child node is at level 1, its grandchild is at level 2, and so on.
        - Keys

- Represents a value of a node based on which a search operation is carried out for a node
- <u>Binary tree</u>
  - A nodes left child must have a value less than its parents value and the nodes right child must have a value greater than its parent value
  - Each nose has at most 2 children
    - Normally named left and right child

- Heap
  - A complete binary tree, meaning it is always balanced
    - This just means that elements on higher levels are greater (for max-heap) or smaller (min-heap) than elements on lower levels
      - Whereas a tree guarantees order (from left to right)
  - Heap is better at finding min and max (O(1)) while a tree is better for finding (O(logN))

Sorting Algorithms

- Bubble sort
  - Repeatedly swapping the adjacent elements is they are in the wrong order
    - Not suitable for large data sets as its average and worst-case time complexity is quite high
  - Time complexity:
    - O(N^2)
  - Auxiliary space:
    - O(1)
  - It can be optimized by stopping the algorithm if the inner loop didn’t cause any swap
- Selection sort
  - Works by repeatedly selecting the smallest (or largest element) from the unsorted portion of the list and moving it to the sorted portion of the list
  - This algorithm maintairs two subarrays in a given array
    - Sorted
    - Unsorted
  - In every interaction of the selected sort, the minimum element from the unsorted subarray is picked and moved to the beginning of the unsorted array
  - Time complexity:
    - O(N^2)
  - Auxiliary space:
    - O(1)
- Merge sort

    - Diving an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array
    - The benefit to this sorting method is that is has a time complexity of O(nlogn) which means it is fast
    - It also is a stable sort which means that the order of elements with equal values is preserved during the sorting process

Class review - C++

- A struct and a class are almost the name and neither requires a typedef
- Use const not #define
- The void in
    - Int foo(void) is deprecated
- For pointers and dynamic allocation
    - Use new and elete
    - Use nullptr not NULL
- #pragma once can be used as a header guard
- Pass by Alias (reference)
    - If the function has variable with &, then it has direct access but without pointer notation

Linked Lists

- A collection of nodes
- There is a pointer called head or front or root that points to the first node
- The last node points to nullptr

- Nodes
    - Traditionally implemented as structs
    - But could be used as a class
        - Specify public access
        - Provide accessor/mutator methods
        - Make the linked-list class a friend of the node class
- Insertion and deletion
    - A pointer that moves down a linked list (referred to as an iterator) is used to insert or delete a node
    - This means that running time for a linked-list method is usually linear
    - The key to insertion and deletion methods in a linked list id to move the iterator to the node before the change
    - Often have to write code that divides into two cases depending on whether the head pointer needs to change or not
- Doubly linked lists

    - Insertion and deletion now have 4 possible cases
        - Neither first and last pointer change
        - First pointer needs to change
        - Last pointer needs to change
        - Insertion when empty or deletion of only node:
            - Both first and last pointer change
    - To reduce insertion and deletion to a single case, have dummy nodes for the first and last node
        - Head and Tail

Functions for Classes

- Copy constructor and the rule of three used to create a copy of an object
- Stream insertion operation:
    - Used to print out object
- Destructor
    - Used to free up memory

Copy Constructor

- Passing by value requires making a new object that is a clone
    - void foobar(Pear ob)
- Then ob is initialized by running the copy constructor
- The compiler inserts a default copy constructor if you dont
    - You should provide one specifically if youre using pointers

Shallow Copy

- Suppose class pear contains pointer to a Banana. If we just use the default copy constructor, then we do
    - Pear P;
    - Pear Q(P);
- The object O will have a pointer that points to the same banana as the pointer in P
    - Meaning that changes to P also change Q
        - Q is a shallow copy

Deep Copy

- A copy whose properties do not share the same reference

Rule of Three

- There are three functions that either the default is okay for all of them or all three should be coded
    - The copy constructor

    - The copy assignment operator
    - The destructor
    - Move constructor
    - Move assignment operator

Stream insertion operator

- This allows the user to write
    - Pear P
    - cout << P;
- This is achieved by a friend function with signature
    - Ostream & operator<< (........)

Friends

- Another class or function can be made a friend so that it has access to the private variables of this class

Memory Management

- Any variable that is created by declaration is automatically released
- Any object that is created by NEWing (dynamic allocation) must be manually released by the user