

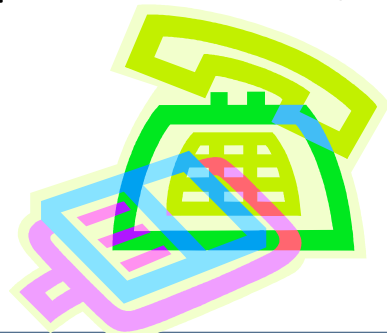
Maps (or Dictionaries)

Goals

- Introduce the Map(or Dictionary) ADT
- Introduce an implementation of the map with a Dynamic Array

So Far....

- Emphasis on ***values*** themselves
 - e.g. store names in an AVL tree to quickly lookup club members
 - e.g. store numbers in an AVL tree for a tree sort
- Often, however, we want to associate something else (ie. a value) with the lookup value (ie. a key)
 - e.g. phonebook, dictionary, student roster, etc.



Map or Dictionary ADT

- A Map stores not just values, but ***Key-Value pairs***

void put (KT key , VT value)

VT get (KT key)

int containsKey(KT key)

void removeKey (KT key)

Struct Association {

KT key;

VT value;

};

All comparisons done on the key

All returned values are VT

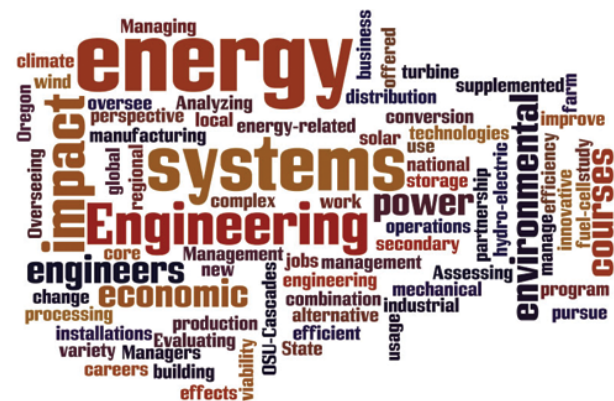
Can implement with AVLTree, HashTable, DynArr,
etc.

Dynamic Array Map: **Contains**

```
int containsMap (DynArr *v, KT key, comparator compare){  
    int i = 0;  
    for (i = 0; i < v->size; i++) {  
        if ((*compare)((struct association *) (v->data[i]))->key,  
            key) == 0 ) /* found it */  
            return 1;  
    }  
    return 0;  
}
```

Map or Dictionary

- A Map stores not just values, but ***Key-Value pairs***
- Example Application: Concordance
 - Count the number of times each word appears in a selection of text (ie. a concordance)
 - Keys: unique words from the text
 - Value: count of each word



Your Turn – Worksheet 36: Dynamic Array Implementation of the Map

- Internally, store **Struct Associations**
- Put
 - Ensure that each element in the dictionary has a ***unique key***
- ContainsKey
 - Loop until you find the 'key' and then return true, else false
- Get
 - Loop until you find the 'key' then return the value
- RemoveKey
 - Loop until you find the 'key', then remove the entire association