1. A researcher wishes to calculate some statistical properties for a collection of integer data values. The data values are represented by the array `tally`. The indexes of the array represent the possible values of the actual data values from zero to the maximal value (15 in the example below). Each array location contains the frequency (number of occurrences) of the value corresponding to its index. In the example below, `tally[4]` is 10, which means that the value **4** occurs ten times in the collection of data; whereas `tally[8]` is 0, which means that the value **8** does not occur in the data collection.

`tally`

Value	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Frequency	0	0	10	5	10	0	7	1	0	6	0	10	3	0	0	1

(a) You will write the function `CalculateModes`, which is described as follows. `CalculateModes` returns an array containing the mode(s) found in parameter `tally`. The length of the returned array is equal to the number of modes.

A **mode** is defined as a value that occurs with maximal frequency. If there is more than one such value, each is considered a mode of the data. In the example above, the modes are 2, 4, and 11, because they each occur 10 times and all other values occur fewer than 10 times.

The following function, `FindMax`, is available for your use. It returns the maximum value in array `nums`. Using the example array, `FindMax(tally)` returns 10.

```
int FindMax(const apvector<int> & nums);
// precondition:  nums.length() > 0
// postcondition: returns the maximum value in nums
```

Do NOT write the body of `FindMax`.

In writing `CalculateModes`, you may call `FindMax` as specified above.

Complete function `CalculateModes` below.

```
apvector<int> CalculateModes(const apvector<int> & tally)
// precondition:  tally.length() > 0
// postcondition: returns an apvector that contains the mode(s);
//                the apvector’s length equals the number of modes
```

GO ON TO THE NEXT PAGE.

(b) You will write the function `KthDataValue`, which is described as follows. `KthDataValue` returns the kth data value when the data values are considered in sorted order. Recall that the indexes of the array represent possible data values and that each array location contains the frequency of the value corresponding to its index.

In the example reprinted below, the first ten data values are **2**, the next five data values are **3**, and the next ten data values are **4**. `KthDataValue(tally, 1)` returns **2**, `KthDataValue(tally, 14)` returns **3**, `KthDataValue(tally, 15)` returns **3**, and `KthDataValue(tally, 16)` returns **4**.

<u>`tally`</u>

Value	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Frequency	0	0	10	5	10	0	7	1	0	6	0	10	3	0	0	1

Complete function `KthDataValue` below.

```
int KthDataValue(const apvector<int> & tally, int k)
// precondition:  tally.length() > 0;
//                0 < k ≤ total number of values in the data collection
// postcondition: returns the kth value in the data collection
//                represented by tally
```

GO ON TO THE NEXT PAGE.

2. Consider the following declaration that will be used to keep track of information about items in a grocery store . Each item is identified by a unique one-word name and has an associated price, size, and category.

```
class GroceryStore
{
  public:
    GroceryStore();

    // modifier

    void SetPrice(const apstring & itemName, double price);
       // changes the price of item associated with itemName

    // accessors

    double GetPrice(const apstring & itemName) const;
       // returns the price of this item

    int GetSize(const apstring & itemName) const;
       // returns the size (in ounces) of this item

    apvector<apstring> GetItems(char category) const;
       // returns a vector (possibly empty) of the names of all
       // items in the specified category

    // ... other public and private members not shown
};
```

(a) You will write free function `ChangePrices`, which is described as follows. `ChangePrices` reads item names and prices from `input` and changes the prices of the corresponding items in `store` to the new prices.

For example, assume `store` contains the following items.

Name	**Price**	**Size (in ounces)**	**Category**
avocado	1.68	8	P
milk	1.92	64	D
chicken	4.48	64	M
broccoli	1.92	16	P
yogurt	0.96	16	D
spinach	1.76	16	P
cornedbeef	6.72	48	M
porkchops	2.24	32	M

Assume that the stream `input` contains the following data.

```
cornedbeef  7.99
yogurt       .75
milk        1.25
broccoli     .98
```

The call `ChangePrices(store, input)` will change the prices of `cornedbeef`, `yogurt`, `milk`, and `broccoli` to the corresponding new prices.

GO ON TO THE NEXT PAGE.

Question 1

Part A: CalculateModes **5 points**

+1 loop over tally
- **+1/2** attempt (must compare tally entries to some value in body of loop)
- **+1/2** correct

+1/2 correct call to FindMax (or correct reimplementation)

+2 initialize and add modes to result vector
- **+1/2** identify and process mode (uses tally[val] in conditional and val in assignment)
- **+1/2** attempts to add mode-thingy to result-thingy (not tally)
- **+1/2** assigns to correct location
- **+1/2** result vector size has been adjusted based on number of modes

+1 initialize, maintain, and use count
- **+1/2** attempt (must attempt to maintain number of modes found with count variable or vector length)
- **+1/2** correct

+1/2 return result vector (must complete loop before return)

Part B: KthDataValue **4 points**

+1/2 initialize counter (in context of incrementing count, get this point if k used to count down)

+1 loop over tally
- **+1/2** attempt (needs reference to tally in body of loop)
- **+1/2** correct (upper bound of loop can be larger than tally length – precondition)

+1 increment counter correctly (or decrement k countdown)

+1 locate correct value (correct exit from while, break from for – if)
- **+1/2** attempt
- **+1/2** correct (lose for OBOB)

+1/2 return value found (must locate value before return)