3. A `WordCollection`, shown in the class declaration below, stores a group of words. The collection may store multiple instances of any word. In this question, you will not implement any of the member functions of class `WordCollection`.

```
class WordCollection
{
  public:
    int Size() const;
       // returns the total number of items stored in the collection

    void Insert(const apstring & word);
       // adds word to the collection (duplicates allowed)

    void Remove(const apstring & word);
       // removes one instance of word from the collection if word is
       // present; otherwise, does nothing

    apstring FindKth(int k) const;
       // returns kth word in alphabetical order, where
       // 1 ≤ k ≤ Size()

    // other public member functions not shown

  private:
    // private data members not shown
};
```

The public member function `FindKth` returns the *k*th word in alphabetical order from the collection (the word with rank *k*), even though the underlying implementation of `WordCollection` may not be sorted. The rank ranges from 1 (first in alphabetical order) to *N*, where *N* is the number of words in the collection. For example, assume that `WordCollection C` stores the following words.

```
{ "at", "bad", "all", "at" }
```

The following table illustrates the results of calling `C.FindKth(k)`.

| k | C.FindKth(k) |
|---|---|
| 1 | "all" |
| 2 | "at" |
| 3 | "at" |
| 4 | "bad" |

(a) Write free function `Occurrences`, as started below. `Occurrences` returns the number of times that `word` appears in `WordCollection C`. If `word` is not in `C`, `Occurrences` should return 0.

In writing `Occurrences`, you may call any of the member functions of the `WordCollection` class. Assume that the member functions work as specified.

Complete function `Occurrences` below.

```
int Occurrences(const WordCollection & C, const apstring & word)
// postcondition: returns the number of occurrences of word in C
```

GO ON TO THE NEXT PAGE.

(b) Write free function `RemoveDuplicates`, as started below. `RemoveDuplicates` removes all but one occurrence of `word` from `C`. If `word` is not in collection `C`, then `RemoveDuplicates` does nothing.

In writing `RemoveDuplicates`, you may call function `Occurrences` specified in part (a). Assume that `Occurrences` works as specified, regardless of what you wrote in part (a).

Complete function `RemoveDuplicates` below.

```
void RemoveDuplicates(WordCollection & C, const apstring & word)
// postcondition: if word is present in C, all but one occurrence
//                is removed; otherwise, C is unchanged
```

(c) Write free function `MostCommon`, as started below. `MostCommon` returns the word that appears most often in the collection. If there is more than one such word, return any one of them. You may assume that `C` is not empty.

In writing `MostCommon`, you may call function `Occurrences` specified in part (a). Assume that `Occurrences` works as specified, regardless of what you wrote in part (a).

Complete function `MostCommon` below.

```
apstring MostCommon(const WordCollection & C)
// precondition:  C is not empty
// postcondition: returns the word that appears most often in C;
//                if there is more than one such word,
//                returns any one of those words
```

**GO ON TO THE NEXT PAGE.**