2. This question involves reasoning about the code from the Large Integer Case Study. A copy of the code is provided as part of this exam.

(a) Write the new `BigInt` public member function `IsOdd`, as started below. `IsOdd` should return `true` if the `BigInt` is odd; otherwise, it should return `false`.

You may NOT assume that the `%` or `%=` operators have been defined for the `BigInt` class.

Complete function `IsOdd` below.

```
bool BigInt::IsOdd() const
// postcondition: returns true if this BigInt is odd;
//                otherwise, returns false
```

(b) Write the free function `Power`, as started below. `Power` returns the value of `base` to the `exp` power, that is $\texttt{base}^{\texttt{exp}}$, where $\texttt{exp} \geq 0$. For example, the call `Power(3, 5)` returns `243`, which is $\texttt{3}^{5}$.

You must use the following algorithm.

```
Initialize a variable, product, to be 1.
While exp is not zero do the following:
    if exp is odd, product is set to product times the base
    square the base
    divide exp by two
When done, product contains the result.
```

Assume that a new member function, `DivBy2`, has been defined for the `BigInt` class, as specified below. `DivBy2` divides this `BigInt` by `2` (using integer division). (You do not need to write the body of `DivBy2`.)

```
void BigInt::DivBy2();  // this BigInt is divided by 2
```

In writing `Power`, you may use the `BigInt` public member function `DivBy2` specified above and you may use the `BigInt` public member function `IsOdd` specified in part (a). Assume that `IsOdd` works as specified, regardless of what you wrote in part (a).

Complete function `Power` below.

```
BigInt Power(const BigInt & base, const BigInt & exp)
// precondition:  base > 0 and exp ≥ 0
// postcondition: returns the value of base to the exp
```

GO ON TO THE NEXT PAGE.

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.