1. A *mode* is a value in an array that is larger than both the value immediately before it in the array and the value immediately after it. In other words, a mode occurs at index `k` in the array `A` if `A[k] > A[k - 1]` and `A[k] > A[k + 1]`. The array is *unimodal* if the values increase until they reach a mode, then decrease, so that there is only one mode. For example, the array `A` shown below is unimodal with its mode occurring at index `4`. Assume that the mode does not occur at the first or last entry in the array.

Index `k`	`A[k]`	
0	3	
1	5	
2	9	
3	10	
4	12	← mode
5	11	
6	9	
7	4	

(a) Write function `IsMode`, as started below. `IsMode` returns `true` if `data[k]` is larger than `data[k - 1]` and larger than `data[k + 1]`; otherwise, it returns `false`. In the example above, the call `IsMode(A, 4)` returns `true` and the call `IsMode(A, 5)` returns `false`.

Complete function `IsMode` below.

```
bool IsMode(const apvector<int> & data, int k)
// precondition:  0 < k < data.length() - 1
```

(b) Write function `ModeIndex`, as started below. `ModeIndex` returns the index of the mode of `data`. You may assume that `data` is unimodal and the mode occurs at an index `k`, where `0 < k < data.length() - 1`. In the example above, the call `ModeIndex(A)` returns `4`.

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

Complete function `ModeIndex` below.

```
int ModeIndex(const apvector<int> & data)
// precondition:  data is unimodal and data.length() ≥ 3
```

GO ON TO THE NEXT PAGE.

(c) Write function `PrintHistogram`, as started below. `PrintHistogram` prints a character histogram of a unimodal array of nonnegative values, `data`, such that the longest bar of the histogram (the mode) has `longestBar` characters `barChar`, and all other bars have a number of `barChar` characters proportional to the corresponding value in the array `data` (rounding down).

For example, assume that `apvector data` contains the values shown below.

The call `PrintHistogram(data, 20, 'x')` will print the histogram shown in the Output column below.

Index `k`	`data[k]`	Length of bar	Output of call `PrintHistogram (data, 20, 'x')`
0	3	5	`xxxxx`
1	5	8	`xxxxxxxx`
2	9	15	`xxxxxxxxxxxxxxx`
3	10	16	`xxxxxxxxxxxxxxxx`
4	12	20	`xxxxxxxxxxxxxxxxxxxx`
5	11	18	`xxxxxxxxxxxxxxxxxx`
6	9	15	`xxxxxxxxxxxxxxx`
7	4	6	`xxxxxx`

In writing `PrintHistogram`, you may call functions `IsMode` and `ModeIndex` specified in parts (a) and (b). Assume that `IsMode` and `ModeIndex` work as specified, regardless of what you wrote in parts (a) and (b).

Complete function `PrintHistogram` below.

```
void PrintHistogram(const apvector<int> & data,
                    int longestBar, char barChar)
// precondition:  data is unimodal and data.length() ≥ 3;
//                data[k] ≥ 0 for 0 ≤ k < data.length()
```

GO ON TO THE NEXT PAGE.

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 `exp` ≥ `0`. For example, the call `Power(3, 5)` returns `243`, which is 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.

2000 AP® Computer Science

A Question 1

Part A: `IsMode` **2 pts**

+1 attempt (needs at least one on task comparison, `>=` is OK, `||` instead of `&&`)

+1 correct (1 for true and 0 for false is OK)

Note: loop that bears no relation to `k` or that destroys `k` gets <u>no points</u>

Part B: `ModeIndex` **2 pts**

+1 search array
- **+1/2** attempt
- **+1/2** correct (note: any length bound `> data.length()` or no length bound works)

+1 identify and return mode index
- **+1/2** attempt (calls `IsMode` or reimplements it – reimpl must be <u>perfect</u> to get attempt)
- **+1/2** correct

Note: `IsMode` function used as `void` function loses the full identify mode index point
Note: Without a loop, `IsMode` must be called correctly to earn mode index attempt ½ point

Part C: `PrintHistogram` **5 pts**

+1 get value of mode
- **+1/2** attempt (must attempt to find mode index before printing anything)
- **+1/2** correct
  (`k = ModeIndex(data)` loses correct if `data[k]` is not used later in the computation)

+1 scan array
- **+1/2** attempt (must have attempt to scan the data and draw a bar in loop)
- **+1/2** correct

+2 compute correct bar length
- **+1** attempt (must use mode value and `longestBar`)
  (In the absence of a mode value, must use a data element and `longestBar`)
- **+1** correct

+1 draw bar
- **+1/2** attempt (must have a loop)
- **+1/2** correct (must use `barChar` and include `endl`)

NOTE: If `A[k]` is used instead of `data[k]`, it is –1/2 usage, confused id