1. A gas station needs to keep track of the number of gallons of gas it has on hand and the current price per gallon. The station has at least two pumps. Pumps 0 and 1 are full-service pumps, and all the rest are self-service. Self-service customers pay the base price for each gallon of gas, while full-service customers pay $0.25 more per gallon.

   Two classes are used to represent this situation. The `Pump` class handles the details for each pump and the `Station` class handles the overall gas station functions.

   Consider the following class declarations.

```
class Pump
{
  public:

    Pump();
    // postcondition: sets number of gallons sold at this pump to 0.0

    double GallonsSold() const;
    // postcondition: returns the number of gallons sold at this pump

    void ResetGallonsSold();
    // postcondition: resets number of gallons sold at this pump to 0.0

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

class Station
{
  public:
    // constructor not shown

    double TotalSales() const;
    // postcondition: returns the total cash value of
    //                sales for all pumps

    void ResetAll();
    // postcondition: for every Pump p in this station
    //                p.GallonsSold() is 0.0

    void CloseStation(ostream & logFile);
    // precondition:  logFile is open and ready for writing
    // postcondition: writes the total cash value of
    //                all gas sold for the day to logFile;
    //                for every Pump p in this station
    //                p.GallonsSold() is 0.0

    // ... other public member functions not shown

  private:
    double myBasePrice;      // current price per gallon of gas
                             // for self-service pumps

    apvector<Pump> myPumps; // the gas pumps; myPumps.length() > 1 and
                             // is the number of pumps in this station

    // ... other private data not shown
};
```

GO ON TO THE NEXT PAGE.

(a) Write the `Station` member function `ResetAll`, as started below. `ResetAll` changes the number of gallons sold at each pump to 0.0.

In writing `ResetAll`, you may call any of the public member functions of the `Pump` and `Station` classes. Assume that all these functions work as specified.

Complete function `ResetAll` below.

```
void Station::ResetAll()
// postcondition: For every Pump p in this station
//                p.GallonsSold() is 0.0
```

(b) Write the `Station` member function `TotalSales`, as started below. `TotalSales` returns the total cash value of the gallons sold at all pumps. Recall that self-service pumps charge the base price for each gallon of gas, while full-service pumps (pumps 0 and 1) charge $0.25 more per gallon.

In writing `TotalSales`, you may call any of the public member functions of the `Pump` and `Station` classes. Assume that all these functions work as specified.

Complete function `TotalSales` below.

```
double Station::TotalSales() const
// postcondition: returns the total cash value of sales
//                for all pumps
```

(c) Write the `Station` member function `CloseStation`, as started below. `CloseStation` will write the total amount of money earned from the day's gas sales to the output stream, `logFile`, and then reset the number of gallons sold at each individual pump to 0.0.

In writing `CloseStation`, you may call any of the public member functions of the `Pump` and `Station` classes. Assume that these functions, including `ResetAll` and `TotalSales`, work as specified, regardless of what you wrote in parts (a) and (b).

Complete function `CloseStation` below.

```
void Station::CloseStation(ostream & logFile)
// precondition:  logFile is open and ready for writing
// postcondition: writes the total cash value of
//                all gas sold for the day to logFile;
//                for every Pump p in this station
//                p.GallonsSold() is 0.0
```

GO ON TO THE NEXT PAGE.

2. Consider the following declarations for maintaining a list of books. Information about each book includes the title, author, and an appropriate age range for readers. The list is ordered by age range, as defined by function `LessThan`. Assume that a book appears at most once in the list.

```
struct Book
{
  apstring title;  // title of book
  apstring author; // author of book
  int lowAge;      // lowest recommended age
  int highAge;     // highest recommended age
};

bool LessThan(const Book & lhs, const Book & rhs);
// postcondition: returns true if lowAge of lhs < lowAge of rhs or
//                if lowAge of lhs and rhs are equal
//                  and highAge of lhs < highAge of rhs;
//                otherwise, returns false

class BookList
{
  public:

    BookList();    // constructor

    void InsertOne(const Book & bk);
    // precondition:  this BookList is in sorted order by age range
    //                as defined by LessThan;
    //                bk is not already in this BookList
    // postcondition: bk has been inserted into this BookList,
    //                maintaining its order by age range

    void InsertMany(const apvector<Book> & second);
    // precondition:  this BookList is in sorted order by age range
    //                as defined by LessThan; second contains
    //                second.length() books in arbitrary order;
    //                none of the books in second are in this BookList
    // postcondition: all the books from second have been inserted into
    //                this BookList, maintaining its order by age range

    // ... other public member functions not shown

  private:

    apvector<Book> myList;
       // collection of books in sorted order as defined by LessThan;
       // myList.length() > 0

    int myCount;
       // number of books in myList
};
```

GO ON TO THE NEXT PAGE.

AP® Computer Science A
2001 SCORING GUIDELINES

Question 1

Part A: `ResetAll` **2 points**

- **+1** loop over pumps
  - **+1/2** attempt — must have some reference to pumps
  - **+1/2** correct
- **+1** reset each pump
  - **+1/2** attempt — must attempt indexing
  - **+1/2** correct

Part B: `TotalSales` **5 points**

- **+1** initialize/return total
- **+1/2** init — correct (`int` total, reset in loop lose this)
  - **+1/2** return (premature return, `cout` lose this)

- **+2** handle full service pumps
  - **+1** attempt — must have two different actions
  - **+1** correct (watch for * 1.25)

- **+2** general case loop
  - **+1** loop
    - **+1/2** attempt
    - **+1/2** correct
  - **+1** accumulate pump sales
    - **+1/2** attempt — must use indexing here (gallons only okay)
    - **+1/2** correct (premature return and gallons only loses this)

Part C: `CloseStation` **2 points**

- **+1** print total to `logFile` (text, formatting, or lack thereof OK)
  - **+1/2** attempt (total sales OR `logFile`)
  - **+1/2** correct

- **+1** `ResetAll()`
  - **+1/2** attempt
  - **+1/2** correct

Usage

- **-1** `Pump/myPumps/p/Station` confusion

- **-0** `obj.Func` instead of `obj.Func()`