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.

In writing `ChangePrices`, you may call any of the public member functions of the `GroceryStore` class. Assume the member functions work as specified.

Complete free function `ChangePrices` below.

```
void ChangePrices(GroceryStore & store, istream & input)
// precondition:  input is open for reading;
//                each line consists of a valid one word item name
//                and a valid price
// postcondition: changes the prices of items in store using names and
//                new prices from input
```

(b) The unit price of an item is the price per ounce. The table below is repeated from part (a) for your convenience.

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

The unit price of avocado is 1.68 divided by 8, which equals 0.21, and the unit price of spinach is 1.76 divided by 16, which equals 0.11.

You will write free function `BargainItem`, which is described as follows. `BargainItem` returns the name of an item whose unit price is the lowest in the specified category. If there is more than one item with the lowest unit price, any one of these items may be returned. If there are no items in the category, `BargainItem` returns `"none"`.

For example, consider the items and prices listed in the table above. Using this table, the results of three calls to `BargainItem` are shown below.

Function call	Returned value
`BargainItem(store, 'P')`	`spinach`
`BargainItem(store, 'M')`	`chicken` or `porkchops`
`BargainItem(store, 'B')`	`none`

In writing `BargainItem`, you may call any of the public member functions of the `GroceryStore` class. Assume that the member functions work as specified.

Complete free function `BargainItem` below.

```
apstring BargainItem(const GroceryStore & store, char category)
// postcondition: returns the name of an item whose unit price
//                is the lowest in the specified category;
//                if no items in the specified category, returns "none"
```

3. This question involves reasoning about the code from the Marine Biology Case Study. A copy of the code is provided as part of this exam.

   Consider modifying fish so they have a direction based on their last move, and move only in a forward direction: either straight ahead or diagonally ahead to the right or left. You will be asked to write three functions for this question:

   (a) a `Position` member function that returns the position to the northeast of the current position,

   (b) a `Fish` member function that returns a neighborhood of positions representing the forward moves that a fish can make, and

   (c) a `Position` member function that returns the direction from this `Position` to an adjacent `Position`.

   For this question, Potential Movement Locations are positions that are:

   (1) adjacent to the current fish position,
   (2) in the direction the fish is moving, `myDir`, or 45 degrees to the right or left of `myDir`, and
   (3) empty and in the environment `env`.

   The diagram below shows three fish, represented by arrows showing their current directions. The Potential Movement Locations for each fish are shaded. For example, for the fish at position (2,1) in the diagram with `myDir` equal `"E"` (as indicated by the arrow), Potential Movement Locations would be those positions to the northeast, east, and southeast that are empty. For the fish at position (2,9) with `myDir` equal `"SW"`, Potential Movement Locations would be those positions to the west, southwest and south that are empty.

Potential Movement Locations

	0	1	2	3	4	5	6	7	8	9	10	11
0					NW	N	NE					
1	NW	N	NE		W	↗	E		NW	N	NE	
2	W	→	E		SW	S	SE		W	↙	E	
3	SW	S	SE						SW	S	SE	
4												

AP® COMPUTER SCIENCE A
2002 SCORING GUIDELINES

Question 2

Part A: ChangePrices **3 points**

+1 loop until the end of the stream that is used for input

+1 input name and price
- **+1/2** attempt (must attempt to read more than one value in the loop, cin or other stream OK for attempt)
- **+1/2** correct

+1 set price
- **+1/2** attempt (must call `store.SetPrice`)
- **+1/2** correct

Part B: BargainItem **6 points**

+1 get names in category
- **+1/2** attempt (must call `store.GetItems`)
- **+1/2** correct

+1 none in category, return "none"
- **+1/2** correct test
- **+1/2** correct return

+1 loop over names in category
- **+1/2** attempt (must have loop that accesses array returned by `GetItems`)
- **+1/2** correct

+1 calculate unit price for item
- **+1/2** attempt (must have a ratio of price and/or size values)
- **+1/2** correct

+1/2 initialize all state variables (min unit price, index, name or a combination)
(if min unit price is initialized with incorrect ratio calculation, deduct under "calculate unit price for item" correct ½ point)

+1 maintain state of search (min item name, index of min item, or min unit price)
- **+1/2** attempt (must adjust some state variable in context of if statement)
- **+1/2** correct

+1/2 return name of best bargain found by search
(may get this point in context of search that is not correct, but must return an item name)