C++ PRIMER PLUS, 5th EDITION
PROGRAMMING EXERCISES
CHAPTER 10

1. Provide method definitions for the class described in Review Question 5 and write a short program that illustrates all the features.

2. Here is a rather simple class definition:

```
class Person {
private:
    static const int LIMIT = 25;
    string lname;              // Person's last name
    char fname[LIMIT];         // Person's first name
public:
    Person() {lname = ""; fname[0] = '\0'; } // #1
    Person(const string & ln, const char * fn = "Heyyou"); // #2
    // the following methods display lname and fname
    void Show() const;         // firstname lastname format
    void FormalShow() const;   // lastname, firstname format
};
```

Write a program that completes the implementation by providing code for the undefined methods. The program in which you use the class should also use the three possible constructor calls (no arguments, one argument, and two arguments) and the two display methods. Here's an example that uses the constructors and methods:

```
Person one;                        // use default constructor
Person two("Smythecraft");         // use #2 with one default argument
Person three("Dimwiddy", "Sam");   // use #2, no defaults
one.Show();
cout << endl;
one.FormalShow();
// etc.  for two and three
```

3. Do Programming Exercise 1 from Chapter 9, but replace the code shown there with an appropriate `golf` class declaration. Replace `setgolf(golf &, const char *, int)` with a constructor with the appropriate argument for providing initial values. Retain the interactive version of `setgolf`, but implement it by using the constructor. (For example, for the code for `setgolf()`, obtain the data, pass the data to the constructor to create a temporary object, and assign the temporary object to the invoking object, which is `*this`.)

4. Do Programming Exercise 4 from Chapter 9, but convert the `Sales` structure and its associated functions to a class and its methods. Replace the `setSales(Sales &, double[], int)` function with a constructor. Implement the interactive `setSales(Sales &)` method by using the constructor. Keep the class within the namespace `SALES`.

5. Consider the following structore declaration:

```
struct customer {
    char fullname[35];
    double payment;
};
```

Write a program that adds and removes customer structures from a stack, represented by a `Stack` class declaration. Each time a customer is removed, his or her payment should be added to a running total, and the running total should be reported. Note: you should be able to use the `Stack` class unaltered; just change the `typedef` declaration so that `Item` is type `customer` instead of `unsigned long`.

6. Here's a class declaration.

```
class Move
{
private:
    double x; double y;
public:
    Move(double a = 0, double b = 0);          // sets x, y to a, b
    void showmove() const;                     // shows current x, y values
    Move add(const Move  m) const;
// this function adds x of m to x of invoking object to get new x,
// adds y of m to y of invoking object to get new y, creates a new
// move object initialized to new x, y values and returns it
    void reset(double a = 0, double b = 0);    // resets x, y to a, b
};
```

Create member function definitions and a program that exercises the class.

7. A Betelguesean plorg has these properties:

Data

A plorg has a name with no more than 19 letters.

A plorg has a contentment index (CI), which is an integer

Operations

A new plorg starts out with a name and a CI of 50.

A plorg's CI can change.

A plorg can report its name and CI.

The default plorg has the name "Plorga". [sic]

Write a `Plorg` class declaration (including data members and member function prototypes) that represents a plorg. Write the function definitions for the member functions. Write a short program that demonstrates all the features of the `Plorg` class.

8. You can describe a simple list as follows:

- The simple list can hold zero or more items of some particular type.
- You can create an empty list.
- You can add items to the list.
- You can determine whether the list is empty.
- You can determine whether the list is full.
- You can visit each item in the list and perform some action on it.

As you can see, the list really is simple; it doesn't allow insertion or deletion, for example.

Design a `List` class to represent this abstract type. You should provide a `list.h` header file with the class declaration and a `list.cpp` file with the class method implementations. You should also create a short program that utilizes your design.

The main reason for keeping the list specification simple is to simplify this programming exercise. You can implement the list as an array, or, if you're familiar with the data type, as a linked list. But the public interface should not depend on your choice. That is, the public interface should not have array indices, pointers to nodes, and so on. It should be expressed in general concepts of creating a list, adding an item to the list, and so on. The usual way to handle visiting each item and performing an action is to use a function that takes a function pointer as an argument:

```
void visit(void (*pf)(Item &));
```

Here `pf` points to a function (not a member function) that takes a reference to `Item` argument, where `Item` is the type for items in the list. The `visit()` function applies this function to each item in the list. You can use the `Stack` class as a general guide.