C++ PRIMER PLUS, 5th EDITION
PROGRAMMING EXERCISES
CHAPTER 9

1. Here is a header file:

```
// golf.h -- for pe9-1.cpp

const int Len = 40;
struct golf
{
    char fullname[Len];
    int handicap;
};

// non-interactive version:
// function sets golf structure to provided name, handicap
// using values passed as arguments to the function
void setgolf(golf  g, const char * name, int hc);

// interactive version:
// function solicits name and handicap from user
// and sets the members of g to teh values entered
// returns 1 if name is entered, 0 if name is empty string
int setgolf(golf  g);

// function resets handicap to new value
void handicap(golf  g, int hc);

// function displays contents of golf structure
void showgolf(const golf  g);
```

Note that `setgolf()` is overloaded. Using the first version of `setgolf()` would look like this:

```
golf ann;
setgolf(ann, "Ann Birdfree", 24);
```

The function call provides the information that's stored in the `ann` structure. Using the second version of `setgolf()` would look like this:

```
golf andy;
setgolf(andy);
```

The function would prompt the user to enter the name and handicap and store them in the `andy` structure. This function could (but doesn't need to) use the first version internally.

Put together a multifile program based on this header. One file, named `golf.cpp`, should provide suitable function definitions to match the prototypes in the header file. A second file should contain `main()` and demonstrate all the features of the prototyped functions. For example, a loop should solicit input for an array of golf structures and terminate when teh array is full or the user enters an empty string for the golfer's name. The `main()` function should use only the prototyped functions to access the golf structures.

2. Redo Listing 9.8, replacing the character array with a `string` object. The program should no longer have to check wheter the input string fits, and it can compare the input string to "" to check for an empty line.

3. Begin with the following structure declaration:

```
struct chaff
{
    char dross[20];
    int slag;
};
```

Write a program that uses placement `new` to place an array of two such structures in a buffer. Then assign values to the structure members (remembering to use `strcpy()` for the `char` array) and use a loop to display the contents. Option 1 is to use a static array, like that in Listing 9.9, for the buffer. Option 2 is to use regular `new` to allocate the buffer.

4. Write a three-file program based on the following namespace:

```
namespace SALES
{
    const int QUARTERS = 4;
    struct Sales
    {       double sales[QUARTERS];
        double average;
        double max;
        double min;
    };

    // copies the lessoer of 4 or n items from the array ar
    // to the sales member of s and computes and stores the
    // average, maximum, and minimum values of the entered items;
    // remaining elements of sales, if any, set to 0
    void setSales(Sales  s, const double ar[], int n);

    // gathers sales for 4 quarters interactively, stores them
    // in the sales member of s and computes and stores the
    // average, maximum, and minimum values
    void setSales(Sales  s);

    // display all information in structure s
    void showSales(const Sales  s);
}
```

The first file should be a header file that contains the namespace. The second file should be a source code file that extends the namespace to provide definitions for the three prototyped functions. The third file should declare two `Sales` objects. It should use the interactive version of `setSales()` to provide values for one structure and the non-interactive version of `setSales()` to provide values for the second structure. It should display the contents of both structures by using `showSales()`.