C++ Primer Plus, 5^{th} *Edition* by Stephen Prata
Chapter 12: Classes and Dynamic Memory Allocation
Review Questions

1. Suppose a `String` class has the following private members:

```
class String
{
private:
    char * str;   // point to string allocated by new
    int len;      // holds length of string
//...
};
```

   a. What's wrong with this default constructor?

```
String::String() {}
```

   The variables `str` *and* `len` *are not initialized. Also, no memory is allocated dynamically, so if the destructor includes the* `delete` *command, the program will have memory problems since freeing unallocated memory has undefined behavior.*

   b. What's wrong with this constructor?

```
String::String(const char * s)
{
    str = s;
    len = strlen(s);
}
```

   `str` *is a pointer to a* `char` *and the actual argument to the function is passed by value, which means that* `str` *is set to point to an automatic variable's memory location. When the function terminates,* `str` *will be a dangling pointer. Also, the class specified that* `str` *would point to dynamic memory, which is not the case here.*

   c. What's wrong with this constructor?

```
String::String(const char * s)
{
    strcpy(str, s);
    len = strlen(s);
}
```

   We never defined the memory location of `str`*. When we use* `strcpy()`*, we are copying the string to a pointer that was never initialized. Also, the class specified that* `str` *would point to dynamic memory, which is not the case here.*

2. Name three problems that may arise if you define a class in which a pointer member is initialized by using `new`. Indicate how they can be remedied.
    (1) *If the memory is not deallocated then there will be a memory leak. To remedy this, you must make sure to include the* `delete` *in the destructor.*
    (2) *The default copy constructor will copy member by member and when the temporary object is destroyed, it will free the memory you previously allocated to another object. To remedy this, you should explicitly define the copy constructor so that there is deep copying*
    (3) *The default assigment operator does not do deep copying and will cause memory problems when the destructor is called for both objects. To remedy this, you should explicitly overload the assignment operator to allow for deep copying.*

3. What class methods does the complier generate automatically if you don't provide them explicitly? Describe how these implicitly generated functions behave.
    The compiler automatically generates a default constructor, destructor, copy constructor, and an overloaded assignment operator. The default constructor creates the object but does not initialize any variables. The default destructor destroys the object but does nothing else. The default copy constructor creates a temporary object whose member values are exactly those of the object used as an argument. The copy constructor uses the same destructor as the other objects do. The default overloaded assignment operator copies member by member, which is also shallow copying.

4. Identify and correct the errors in the following class declaration:

```
class nifty
{
// data
    char personality[];
    int talents;
// methods
    nifty();
    nifty(char * s);
    ostream & operator << (ostream & os, nifty & n);
}

nifty:nifty()
{
    personality = NULL;
    talents = 0;
}

nifty:nifty(char * s)
{
    personality = new char [strlen(s)];
    personality = s;
    talents = 0;
}

ostream & nifty:operator<<(ostream & os, nifty & n)
{
    os << n;
}
```

Here is what the code should look like:

```
using std::ostream;
class nifty
{
// data
    char * personality;
    int talents;
// methods
    nifty();
    nifty(const char * s);
    friend ostream & operator<<(ostream & os, const nifty & n);
};

nifty::nifty()
{
    personality = NULL;
    talents = 0;
}

nifty::nifty(const char * s)
{
    personality = new char[std::strlen(s) + 1];
    std::strcpy(personality, s);
    talents = 0;
}

ostream & operator<<(ostream & os, const nifty & n)
{
    os << n;
    return os;
}
```

5. Consider the following class declaration:

```
class Golfer
{
private:
    char * fullname; // points to string containing golfer's name
    int games;       // holds number of holf games played
    int * scores;    // points to first element of array of golf scores
public:
    Golfer();
    Golfer(const char * name, int g = 0);
    // creates empty dynamic array of g elements if g > 0
    Golfer(const Golfer & g);
    ~Golfer();
};
```

a. What class methods would be invoked by each of the following statements?

```
Golfer nancy;                          // #1
Golfer lulu("Little Lulu");            // #2
Golfer roy("Roy Hobbs", 12);           // #3
Golfer * par = new Golfer;             // #4
Golfer next = lulu;                    // #5
Golfer hazzard = "Weed Thwacker";      // #6
*par = nancy;                          // #7
nancy = "Nancy Putter";                // #8
```

Statement 1 would use the default constructor.
Statement 2 would use the explicit constructor where the second argument assumes its default value of 0.
Statement 3 would use the explicit constructor.
Statement 4 uses the default constructor.
Statement 5 uses the copy constructor, and possibly the default assignment operator.
Statement 6 uses the explicit constructor, and the copy constructor.
Statement 7 uses the default overloaded assignment operator.
Statement 8 uses the explicit constructor and the default assignment operator.

b. Clearly, the class requires several more methods to make it useful. What additional method does it require to protect against data corruption?

We need to explicitly define a copy constructor and the assignment operator to ensure deep copying. Also, we must ensure that the destructor deallocates memory with the `delete` *command.*