Chapter 2: Solutions to Sample Exam Questions
Short Answer Questions

2.1. Assuming that we are using the namespace main_savitch_2A:

```
throttle quiz;
quiz.shut_off( );
cout << quiz.flow( );
```

2.2.

```
class throttle
{
public:
    throttle( );              // C
    throttle(int size);       // C
    void shut_off( );
    void shift(int amount);
    double flow( ) const;     // X
    bool is_on( ) const;      // X
    ...
```

2.3. If you write a class with no constructors, then the compiler automatically creates a simple default constructor called the "automatic default constructor". The automatic default constructor doesn't do much work--it just calls the default constructors for the member variables that are objects of some other class.

2.4. An inline member function has it's entire implementation given right in the class definition. For example, the throttle's shut_off member function shown here:

```
class throttle
{
public:
    throttle( );
    throttle(int size);
    void shut_off( ) { position = 0; }
    ...
```

2.5. A macro guard consists of three # compiler directives to avoid the accidental duplicate inclusion of some defintions. For example:

```
#ifndef throttle_H
#define throttle_H
   ...
#endif
```

2.6.

```
#include "throttle.h"
using namespace main_savitch_2A;
or
using namespace main_savitch_2A::throttle;
```

2.7. A const reference parameter guarantees that a function's implementation does not change the value of the parameter. It may be used whenever the function's implementation does not even try to change a parameter (and usually when the parameter might be a large data type rather than something small like an int).
For example:

```
double length(const Point& p)
```

```
        // Postcondition: The return value is the distance of p from the
origin.
        {
            return sqrt(p.get_x( )*p.get_x( ) + p.get_y( )*p.get_y( ));
        }
```

2.8. x should be a value parameter if you do not want the changes to change the actual argument. x should be a reference parameter if you do want the changes to change the actual argument. x could never be a const reference parameter in this example (since the body of the function is changing x).

2.9. Use a const reference parameter when the function's implementation does not even attempt to change the value of the parameter.

2.10. Use a reference parameter when the function's implementation does change the value of a parameter and you want the change to alter the actual argument.

2.11. This is a bit of a trick question, and I'm not sure that I would ask it on an exam. The trick is to realize that even the const member function can alter x.size (although it cannot alter the size member variable of the object that activates the const function).

`x = y;`	Legal in all three places.
`x.size = y.size;`	Not legal in main program, but legal in a member function or a friend function.
`x.size = 3;`	Not legal in main program, but legal in a member function or a friend function.
`x.h(42);`	Legal in all three places.

2.12. `foo foo::operator +(const foo& f1, const foo& f2);` The argument names f1 and f2 are unimportant. Usually f1 and f2 are const reference parameters, but there may be situations where that is not true. The function may also need to be a friend function (if it needs access to the foo private members).

2.13. The operator + must be a friend of the class.

2.14. You are implementing a class and you want to write a non-member function that needs access to the class's private members.

Multiple Choice Questions:

2.1 D	2.6 C	2.11 D
2.2 D	2.7 A	
2.3 D	2.8 D	
2.4 C	2.9 C	