1. For each statement, you should first try to guess what it prints. Then run the program, and compare your answer with the program's execution result. Your written answer should consist of the result of each print statement, and an explanation why it printed the value that it did.

```
   inline void swap( int &a, int &b) { int tmp = a; a = b; b = tmp; }
   inline void messup( int *c) { int save = *c; --c; *c = save; }

   int main() {
       int x = 1, y = 2, z = 3;
     /* 1.1 what does the following print? */
       int *a = &x, *b = &y; cout << "1.1: " << *a << " " << *b << endl;
     /* 1.2 what does the following print? */
       swap(a,b); cout << "1.2: " << a << " " << b << endl;
     /* 1.3 how about this? */
       cout << "1.3: " << x << " " << y << endl;
     /* 1.4 and that? Please also give your opinion on the code */
       messup(&z); cout << "1.4: " << x << " " << y << endl;
   }
```

2. Give the code for a simple class CombinationLock class as in Exercise 2.22. In addition, your lock should keep a state open/close.
Your class should have the public interface:

```
   class CombinationLock {
     // constructor with combination, should make a locked lock
       CombinationLock(int, int, int);
     // no copy assignment nor copy constructor
       void close();
     // open with a combination
       bool open(int, int, int); // returns true if succeeds, otherwise
     // prod the state
       bool is_open();
     // plus some private members
   };
```

Explain the reason why and the mechanism used to disable copying.

3. Make the class Rational (Figures 2.12 and 2.13) a class template. You should assume the numerator and denominator have the same type. Discuss what
has to be changed in the interface of the class (Hints: rewrite line 21 with a template declaration, and add typedefs to the template parameter inside the class), and what is not changed (for instance, none

of the member functions or operators need to be changed).

In addition, list ALL the requirements on your template parameter (along with the member function / figure+page+line number). For instance, the constructor that takes two IntTypeis requires a copy constructor on IntType.

4. What does sizeof(string) return, and why? (Assume we use the class of figure 2.22 on page 76.) Answer exercise 2.19 (a) and (b).

5. STL ALGORITHMS: Fill in the missing arguments in the following program -- the missing arguments are indicated by ...'s. The comments describe what each line is supposed to do. You shouldn't have to add any extra code. (Hint: paste the following into a main() function, and add the necessary include files; your answer should compile and execute without errors or aborting due to a failed assertion.)

```
    // initialize a few arrays
    int A1[6] = { 0, 1, 2, 3, 4, 5 };
    int A2[6] = { 5, 4, 3, 2, 1, 0 };
    int A3[6] = { 3, 0, 3, 1, 1, -2 };

    // V is a vector that contains 5 4 3 2 1 0
    std::vector<int> V( A2, A2+6 );

    // L is a list that contains 3 0 3 1 1 -2
    std::list<int> L;
    L.push_back(3); L.push_back(0); L.push_back(3);
    L.push_back(1); L.push_back(1); L.push_back(-2);

    // S is a string that contains "A man, a plan, a canal: Panama!"
    std::string S( " A man, a plan, a canal: Panama!" );

    // 5.0  count the number of 3's in L
    assert( count( ..., ..., ... ) == 2);
    // As an example, your solution should say
    // assert( count( L.begin(), L.end(), 3 ) == 2);

    // 5.1  using std::find, check that V does not contain 6
    assert( std::find( ..., ..., 6 ) == ... );

    // 5.2  test if L contains two adjacent identical numbers
    list<int>::iterator it1 = std::adjacent_find( ..., ... );
    if (it1 == ...)
      std::cout << "No adjacent number found\n";
```

```
else
  std::cout << "Found two adjacent numbers equal to " << *it1 << "\n";

// 5.3  find first punctuation character in S
std::string punctuation( ":;,.!?" );
std::cout << "First punctuation character in " << S << " is "
          << *(std::find_first_of( ..., ..., ..., ... ))
          << std::endl;

// 5.4  check that L and A3 contain the same elements in same order
assert( std::equal( ..., ..., ... ) );

// 5.5 fill A3 with -1
std::fill( ..., ..., -1 );

// 5.6  replace every space in S by the underscore '_'
std::replace( ..., ..., ..., ... );

// 5.7  check that S is a palindrom by reversing S3
std::string S2( "amanaplanacanalpanama" );
std::string S3( S2 );
std::reverse( ..., ... );
assert( std::equal( S2.begin(), S2.end(), S3.begin()) );

// 5.8  copy every element of A2 into A3, check the result
std::copy( ..., ..., ... );
assert( std::equal( A2, A2+6, A3 ) );

// 5.9  sort A3, check that it gives the same as A1
std::sort( ..., ... );
assert( std::equal( A1, A1+6, A3 ) );

// 5.10 find the smallest element in V
assert( std::min_element( ..., ..., ... ) == &V[5] );

// 5.11 find the 3rd smaller element in V
std::nth_element( V.begin(), V.begin()+2, V.end() );
assert( V[0] < 2 && V[1] < 2 && V[2] == 2 );
```