This worksheet is for your use during and after lecture. It will not be collected or graded, but I think you will find it a useful tool as you learn C++ and study for the exams. Explain all false answers for the "True or False" questions; in general, show enough work and provide enough explanation so that this sheet is a useful pre-exam review. I will be happy to review your answers with you during office-hours, via Email, or instant messaging.

1. Name six binary operators in C++.

   Solution: Addition, subtraction, multiplication, division, modulus, input (<<), or output (>>).

2. When binary operators are overloaded by global functions:

   (a) True or False: Both operands must be of the same class.

   Solution: False.

   (b) True or False: The calling object is on the left of the operation symbol.

   Solution: False. There is no calling object for global operators.

   (c) True or False: The operator should return one of the arguments provided.

   Solution: False. Only the case for << or >>.

   (d) True or False: The operator must be written in some class' scope.

   Solution: False. The function is global, it cannot be in **any** class' scope.

3. (a) Write the protoype of a `friend` global operator for taking the modulus of a `myClass` object (LHS) and an integer (RHS). It should return an integer value.

   Solution: `friend int operator%( const myClass&, int rhs );`

   (b) Where would you find the answer to part a within `myClass`' source files?

   Solution: In `myClass.h`, within the class declaration (required since it is a `friend` function).

   (c) Write the protoype of a global operator for multiplying an integer (LHS) with a `myClass` object (RHS). It should return a new `myClass` object.

   Solution: `myClass operator*( int lhs, const myClass& rhs );`

   (d) Where would you find the answer to part c within `myClass`' source files?

   Solution: In `myClass.h` but outside of the class declaration (required since it is **not** a `friend` function).

   (e) Assume `myClass` has a default constructor. Write a snippet of C++ that would invoke the two global operators prototyped in parts a and c.

   Solution:

```
myClass a;
int b;
int i = a % b;
myClass c = i * a;
```

4. The `friend` keyword may occur in C++ source *only* between the curly braces of what?

   Solution: A class declaration.

5. The following is the **function header** of a global operator:

```
int operator-( const myClass& lhs, int rhs )
```

   Is the operator a `friend` of `myClass` or not?

   A. Certainly not.

   B. Maybe, I could tell for sure if I saw the function implementation.

   C. Maybe, I could tell for sure if I saw the `myClass` declaration.

   Solution: C — the only way to be absolutely sure is to see the `myClass` declaration.

6. Write the prototype of a global `operator<<` for `myClass`.

   Solution: `ostream& operator<<( ostream& os, const myClass& arg );`

7. Write the prototype of a global `operator>>` that is a `friend` of `myClass`. Where is this protoype found?

   Solution: `friend istream& operator>>( istream& is, myClass& arg );`
   The protoype must be in the declaration of `myClass`.

8. Consult the appendix of your Mines Calculus book and review the topic of complex numbers. Write a C++ class representing complex numbers with floating point real and imaginary parts. Implement addition ($z_1 + z_2$), scalar multiplication (αz), and an output operator that prints the complex number in the same format as your mathematics reference does.

Solution:

```
#include <iostream>
using namespace std;
/***
 * A complex number class with operator overloading. We make life simple, and
 * put everything into public so we don't have to write 4 separate accessor
 * functions.
 *
 * PLEASE think about this:
 * 1. There are no values for real and imag that create an INVALID complex
 * number.
 * 2. The complex class doesn't have any data members with DEPENDENCE on any
 * other data members.  There is no *CONSISTENCY* we must maintain!  So we
 * don't need to make .real and .imag private!
 *
 * I have arbitrarily choosen to make operator+ and operator* friend functions.
 **/
class complex {
public:
    double real, imag;
    complex();
    complex( double r, double i );
    friend complex operator+( const complex& lhs, const complex& rhs );
    friend complex operator*( const complex& lhs, const double& rhs );
};

// Global output function --- my math reference uses (,) notation.
// Since everything is public, we don't need to friend this.
ostream& operator<<( ostream& os, const complex& rhs )
{
    return os << "(" << rhs.real << "," << rhs.imag << ")";
}
// For completeness, support alpha*z
complex operator*( const double lhs, const complex& rhs )
{
    // return an anonymous object
    return complex( lhs*rhs.real, lhs*rhs.imag );
}

// ctors
complex::complex( double r, double i )
    : real(r), imag(i)
{ /* empty */ }
complex::complex( )
    : real(0), imag(0)
{ /* empty */ }

// ops
complex operator+( const complex& lhs, const complex& rhs )
{
    // return an anonymous object
    return complex( lhs.real+rhs.real, lhs.imag+rhs.imag );
}
```

```
complex operator*( const complex& lhs, const double& rhs )
{
    // return an anonymous object
    return complex( lhs.real*rhs, lhs.imag*rhs );
}
```