Login name ________________ **Name** ______________________

Quiz 2
CSE 131
Spring 2010

Signature __________________ **Student ID** _________________

1. Check #5 Given the following Reduced-C definitions:

```
function : float foo1( float a )   { int b; return b; }
function : float foo2( float & a ) { int b; return b; }
function : float foo3( int a )     { int b; return b; }

int x;  /* global variables */
float y;
```

For each of the following statements, indicate the type of error (if any) that should be reported according to the Project I spec for this quarter (which is similar to C++ rules). Use the letters associated with the available errors in the box below.

> A) No Error
> B) Arg passed to reference param is not a modifiable L-val
> C) Argument not assignable to value param
> D) Argument not equivalent to reference param

```
y = foo1( y );              ____
y = foo1( &y );             ____
y = foo3( y + x );          ____
y = foo2( foo1( y ) );      ____
y = foo1( x + y );          ____
y = foo1( x );              ____
y = foo3( (int) y );        ____
y = foo1( foo2( y ) );      ____
y = foo1( 42 ):             ____
y = foo2( x );              ____
y = foo2( x + y );          ____
y = foo3( x + y );          ____
y = foo1( (float) x );      ____
y = foo3( x + x );          ____
y = foo2( foo1( y ) );      ____
```

2. Modifiable L-vals, Non-Modifiable L-vals, R-vals

Using the Reduced-C Spec (which closely follows the real C language standard), given the definitions below, indicate whether each expression evaluates to either a

A) Modifiable L-val B) Non-Modifiable L-val C) R-val

```
function : float * foo() { /* Function body not important. */ }
bool[9] a;
int x;
const bool y = true;
structdef S1 { int a; float b; bool c; };
S1 z;
S1 *p = &z;
```

```
____ p->b     ____ z.c        ____ ++*foo()       ____ x++             ____ ++x

____ *foo()   ____ *(int *)p  ____ foo()          ____ 42              ____ (float)x

____ p        ____ a[2]       ____ (float *)&x    ____ *(float *)&x    ____ (int *)foo()

____ *&x      ____ y          ____ x = z.a        ____ *foo() * x      ____ *(int *)&p->b

____ *a       ____ &x         ____ a              ____ y != false      ____ *(int *)foo()
```

3. Choosing from the following five operators,

```
&     =     <=     !=    ||
```

list all the operators that are valid for each of the `__Op#__` below in each Reduced-C statement.

```
const int a = 4 __Op1__ 2;
bool b = 2.5 __Op2 __ 3;
bool c = b __Op3__ (b __Op4__ false);
```

`__Op1__` can be: _____

`__Op2__` can be: _____

`__Op3__` can be: _____

`__Op4__` can be: _____

4. Check #8: How did your group implement handling and reporting arithmetic exceptions in the context of constant folding?

5. The statements below refer to the bindings of names to objects.
Fill in the blanks with either A) dynamic or B) static:

_____ binding generally results in greater efficiency of program execution.

_____ binding allows greater flexibility in program implementation.

_____ scoping determines bindings at run time.

_____ binding occurs at compile/link time.

_____ scoping determines bindings at compile time.

_____ scoping is to space as _____ scoping is to time.