Login name _____ **Quiz 2** **Name** _____

CSE 131

Signature _____ **Spring 2008** **Student ID** _____

1. Phase 0 Scoping Fix. Fill in the blanks of the following Reduced-C program with correct types to test if your fix to the scoping bug present in the starterCode works correctly. If the scoping bug is fixed, this program should compile without error. If the bug is not fixed, this program should generate an assignment error at the line x = y;

```
_______  x;       // global x

function : int main() {

  _______  x;     // local x

  bool y;

  x = y;          // If fixed, this line will not cause an error!
                  // If not fixed, this line will cause an error!
  return 0;
}
```

2. Types, STOs, Modifiable L-vals

Given the following C / C++ code fragment:

```
int x = 420;
float y = 4.20;
int * ptr1 = &x;
int * ptr2 = ptr1;
```

for each statement indicate whether it will cause a compile error or not? Treat each statement individually as if it was the only statement following the definitions above. Remember: the increment operator is performing arithmetic (addition) and assignment. The result of this operation is the incremented value.

A) No Error
B) Compile Error

```
&*ptr2 = ptr1;          _____

ptr1 = *&ptr2;          _____

*&ptr2 = ptr1;          _____

ptr2 = &++x;            _____

(float *)ptr2 = &y;     _____

ptr2 = ++&x;            _____

++*(float *)&x;         _____

ptr1 = &*ptr2;          _____
```

3. Type Inference. Consider the following Reduced-C program:

```
const bool a = 5 _Op1_ 7;
const int b = 5 _Op2_ 7;
const bool c = true _Op3_ false;
float x;
bool z;

function : int main()
{
  if ( a ) { return 1; }

  x = b;
  z = c;
  return 0;
}
```

For **_Op1_**, **_Op2_**, and **_Op3_**, list what operators are valid (i.e., cause no compile errors). The available operators are listed below. Some _Op#_ have more than one possible valid operator.

```
!=   ||   /   <
```

_Op1_: _____

_Op2_: _____

_Op3_: _____

4. Semantic Checks/Errors:
Given the following Reduced-C code fragment:

```
int a;
float b;

function : void foo( int x, float & y )
{ /* function body */ }
```

Using variables `a`, `b`, and the expression `(a + b)` as possible arguments to the function `foo()`

Give an example function call to foo() that triggers an assignability error (and only this error).

Give an example function call to foo() that triggers an addressability error (and only this error).

Give an example function call to foo() that triggers an equivalence error (and only this error).

What four operators in our Reduced-C language result in a modifiable l-val when evaluated in an expression?

_____ _____ _____ _____