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

CSE 131B

Signature _____ **Spring 2006** **Student ID** _____

1. Static vs. Dynamic Scoping. Consider the following C-like program:

```
int x = 0;

int f () { return x; }

int g () { int x = 1; return f(); }

int main()
{
  if ( input() > 5 )   /* input() reads input from the user. */
    output( g() );     /* output() prints out a value. */
  else
    output( f() );
}
```

Fill in the table of what value gets printed based on the user input and whether the language uses static or dynamic scoping.

Input	Static scoping	Dynamic scoping
7		
3		

2. Structural vs. Name Equivalence

Using the following type and variable definitions, define the variables that can satisfy the specified properties. If no such variable can exist, state "does not exist". You may not define other types; use only the types defined below.

```
TYPE rec = RECORD x : INTEGER; END;
TYPE rec2 = rec;
TYPE ptr = POINTER TO INTEGER;
TYPE ptr2 = ptr;
VAR a : rec;
VAR b : ptr;
```

1) Define a variable **c** that is strict name equivalent to a:

2) Define a variable **c** that is loose name equivalent to b, but not strict name equivalent to b:

3) Define a variable **c** that is loose name equivalent to a, but not structural equivalent to a:

4) Define a variable **c** that is structural equivalent to b, but not (loose or strict) name equivalent to b:

3. Type Inference. Consider the following Oberon program:

```
CONST a = 5 _Op1_ 7;
CONST b = 5 _Op2_ 7;
CONST c = TRUE _Op3_ FALSE;
VAR x : INTEGER;
VAR z : BOOLEAN;
BEGIN
  IF ( a ) THEN
    RETURN 1;
  END;
  x := b;
  z := c;
  RETURN 0;
END.
```

For **_Op1_**, **_Op2_**, and **_Op3_**, list what operators are valid (i.e., cause no compile errors). The available operators are listed below. Two ops have two possible operators; one op just one.

```
=    AND    *    >=
```

_Op1_: _____

_Op2_: _____

_Op3_: _____

3. The C language specifies all types use _____ equivalence except for _____.

Briefly state what makes a definition different from a declaration.

What is the only allowable type for a struct/record data member in a recursive type definition for a type named **struct fubar**?

What two operators in our Nano-Oberon project result in a modifiable l-val when evaluated in an expression?

In languages like C++ and Java, an instance method definition has access to an identifier known as **this**.
a) What does **this** refer to?

b) How does **this** get set to what you answered in a) above?