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

CSE 131B

Signature _____ **Winter 2006** **Student ID** _____

1. Semantic Analysis and Separate Compilation. Consider the following two C program files:

```
/* file1.c */
#include <stdio.h>
extern int a;
extern int foo( int z );
static int x = 420;

int main( int argc, char *argv[] ) {
   int i = a;

   for ( i = 0; i < a; ++i )
      (void) printf( "%d ", foo( i ) );
   return 0;
}
```

```
/* file2.c */
#include <stdio.h>
extern int x;

float a = 4.20;

void foo( int y ) {
   static int b = 25;

   ++b;
   (void) printf( "%d ", b );
   (void) printf( "%d ", x );
}
```

Trying to separately compile each file and then link the resulting object modules

gcc –c file1.c	file1.c –> cpp –> ccomp –> as –> file1.o
gcc –c file2.c	file2.c –> cpp –> ccomp –> as –> file2.o
gcc file1.o file2.o	file1.o & file2.o –> ld –> a.out/.exe

results in just one error being reported. We discussed some of the problems/complications imposed on the compiler to be able to perform static semantic type checking with separate compilation.

What error will be reported (specify the symbol name and a general description of what the problem is). Hint: The error will be reported in the 3rd gcc call which attempts to link the already compiled and assembled object modules. Hint Hint: Think scope.

Assuming we fixed this error so the program will fully compile/link. How many times does the variable **b** in function **foo()** get initialized?

Can we change the initialization of **b** in file2.c to be `static int b = y;` Why or why not?

Identify two other potential semantic errors in this program that the C compiler and linker did not detect, but lint will identify.

1)

2)

2. 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;
  END;
  x := b;
  z := c;
END.
```

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

```
            +   OR   #   <=
```

_Op1_: _____

_Op2_: _____

_Op3_: _____

3. Constant Folding. For each of the blanks in the program below, if compile time constant folding can be done for the expression, write the result of the constant folding (i.e. the value -420); write "no" if constant folding cannot be done statically by the compiler.

```
VAR x : INTEGER;
VAR y : INTEGER;
VAR z : INTEGER;

CONST a = 5;
CONST b = a + 5;
CONST c = a + b;

PROCEDURE foo (x : INTEGER);
VAR a, b : INTEGER;
BEGIN
    a := 10;

    z := 1 + 2;              ______
    z := x + 2;              ______
    z := a + b;              ______
    z := x + y;              ______
    z := b + c;              ______
END foo;

BEGIN
    x := 4; y := 5;

    z := 1 + 2;              ______
    z := x + 2;              ______
    z := a + b;              ______
    z := c + y;              ______
    z := b + c;              ______

    foo(420);
END.
```