Signature___________________ **Quiz 5** **Name ______________________**

CSE 131B

Login name _________________ **Spring 2004** **Student ID __________________**

1. State which calling convention / parameter passing mode is being used and what gets printed:

Oberon

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

PROCEDURE foo1 ( a : INTEGER; VAR b : INTEGER );
BEGIN
    a := 69;                      Parameter passing mode for a ________________________
    b := 420;
END foo1;                         Parameter passing mode for b ________________________

BEGIN
    x := 911; y := 404;

    foo1( x, y );
    OUTPUT "x = ", x, "; y = ", y, "\n";         Output: x = _______; y = _______
END.
```

Fill in the blanks of the equivalent C program to simulate the above Oberon parameter passing modes (that exposes what the compiler is actually doing to implement these parameter passing modes):

```
int x, y;

void foo1(_____________ a, _____________ b ) {

  ________ = 69;

  ________ = 420;
}

int main( void ) {
  x = 911; y = 404;

  foo1( _____________ , _____________ );
  printf( "x = %d; y = %d\n", x, y );
  return 0;
}
```

2. What kind of information is potentially dangerous to store in unprotected Stack Frames on the Runtime Stack, especially in languages like C/C++?

3. Given the following code for foo(), <u>write an equivalent more highly optimized version in SPARC assembly</u>.

Assume: x is mapped to local register %l1
y is mapped to local register %l2
z is mapped to local register %l3
b is a global variable allocated in the Data segment and NOT mapped to a register

<u>Oberon</u>

```
VAR b : INTEGER;

PROCEDURE foo( i : INTEGER );
  VAR x, y, z : INTEGER;

BEGIN
  x := 29;
  y := x - 5;
  z := i + y;

  b := z;
  b := b * 256;

END foo;



BEGIN
  foo( 10 );
END.
```

<u>SPARC Assembly</u>

```
        .global  main, foo

        .section ".data"
        .align 4
b:      .word 0

        .section ".text"
foo:                                ! mapping local vars
        save    %sp, -96, %sp       ! to local regs

        mov     29, %l1             ! r1 = 29
        sub     %l1, 5, %l2         ! r2 = r1 - 5
        add     %i0, %l2, %l3       ! r3 = param1 + r2

        set     b, %l4
        st      %l3, [%l4]          ! b = r3
        set     b, %l4
        ld      [%l4], %o0          ! out param1 = b
        set     256, %o1            ! out param2 = 256
        call    .mul                ! b * 256
        nop
        st      %o0, [%l4]          ! b = b * 256

        ret
        restore

main:
        /* Code for main() not important */
```

```
Rewrite only code that is
in the bounds of the
rectangle.
```

What question would you most like to see on the Final?