1. What is the output of the following Reduced-C program:

```c
int c = 5;

function : int foo( int x, int & y )
{
    int z = y;
    --x;
    ++y;
    --z;
    cout << c << endl;
    cout << x << endl;
    cout << y << endl;
    cout << z << endl;
    return y++;
}

function : int main( )
{
    int a = 9;
    int b = 2;
    a = foo( b, c );
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    return 0;
}
```

Fill in the blanks below to simulate the above in C. Basically what is really happening under the code.

```c
int foo( ___________ x, ___________ y )
{
    int z = __________;
    -- _________;
    ++ _________;
    --z;
    printf( "%d\n", _________ );
    printf( "%d\n", _________ );
    printf( "%d\n", _________ );
    printf( "%d\n", z );
    return _________________;
}

int c = 5;

int
main( )
{
    int a = 9;
    int b = 2;
    a = foo( __________, __________ );
    printf( "%d\n", a );
    printf( "%d\n", b );
    printf( "%d\n", c );
    return 0;
}
```

Parameter passing mode for x ________________________

Parameter passing mode for y ________________________
Given the following variable definitions

\[
\begin{align*}
\text{int } & a = -99; \quad \text{// initialized global variable} \\
\text{static int } & b; \quad \text{// uninitialized static variable}
\end{align*}
\]

write the SPARC assembly code which should be generated to properly allocate space for each along with their initial values and alignment and to ensure proper access/visibility to these variables if another file is linked to this code's object file.

```asm
.section _____________
.align _____________
a: __________  -99
        __________  a

.section _____________
.align _____________
b: __________  __________
```

Assume local int * variables a and b are allocated space in a function's stack frame at memory locations

\[
\begin{align*}
\text{int } * & a \quad \%fp-4 \\
\text{int } * & b \quad \%fp-8
\end{align*}
\]

Complete the SPARC assembly instructions for the line

\[
b = a++;
\]

that a Reduced-C compiler from this quarter might emit.

You can assume all the initializations of the local variables have been performed. Just emit the code to perform the expression on the right side of the assignment statement and assign the result into b.

We will need to use a temporary or two on the stack, so we will use location %fp-12 for tmp1 and %fp-16 for tmp2.

Follow the basic ld/ld/compute/st model.

```asm
ld  [_______________], %o0
   _____ %o0, [%fp – 12] ! tmp1 = a

ld  [_______________], %o0
   _____ _____, %o1
add  %o0, %o1, %o0
   _____ %o0, [_______________] ! tmp2 = a + ?

   _____ [_______________], %o0
   _____ %o0, [_______________] ! a = tmp2

st  %o0, [_______________] ! b = tmp1
```