1. Consider the following pseudocode and scoping questions [8pts]:

```c
int x; -- global var declaration
void set_x( int n )
    x = n;
void alpha()
    set_x( 4 );
    cout << x;
void beta()
    int x = 5; -- local var declaration
    set_x( 7 );
    cout << x;
set_x( 0 );
alpha();
cout << x;
beta();
cout << x;
```

What does the program output if the language uses static scoping?

What does the program output if the language uses dynamic scoping?

2. From this quarter’s Reduce-C spec (which follows closely the real C language standard), indicate whether each expression is either a: (A) Modifiable L-val (B) Non-Modifiable L-val (C) R-val [1pt each]

```c
function : float & f1() { /* function body not important. */ }  
function : float * f2() { /* function body not important. */ }  
structdef FOO { int x; };
FOO* foo;
int x;
const int y = 5;
int [5] a;
int *p = &x;

_____ f1()   _____ f2()   _____ *f2()   _____ &f1()
_____ (float *)p   _____ *(float *)p   _____ (float *)&y   _____ *(float *)&y
_____ &x   _____ a   _____ y   _____ x*a[1]   _____ a[2]   _____ *p
_____ 17   _____ x   _____ p   _____ *&x*p   _____ &p   _____ foo->x
```

3. Identify the following C constructs as either: (A) Pure Declaration (B) Definition [1pt each]

```c
_____ int x;   _____ int foo( int x ) { return x; }   _____ float bar( float z );
_____ struct fubar;   _____ extern int * func1( int x, float y );   _____ struct fubar { int x; } s1;
```

4. Give an example of a non-converting type cast (underlying bit pattern does not change) [4pts]:

(over)
5. Using Reduced-C syntax, define a pointer to an array of 8 pointers to float named baz such that the following statements will all be valid expressions. This will take two lines of Reduced-C code. [4pts]

```c
typedef float* F;
float x;
F[8] y;
y[7] = &x;
baz = &y[0];
*(baz)[7] = 1.25;
```

6. In Reduced-C (which follows closely the real C standard) we use _______________ name equivalence with struct types. And we use _______________ name equivalence with typedefs. For all other types we use _______________ equivalence. [4pts each]

7. Semantic Checks based on the following Reduced-C code fragment [3pts each]:

```c
int a;
float b;

function : float & foo(float & x, int 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 addressability error (and only this error).

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 equivalence error (and only this error).

Assume the function body above contains float z = 4.20; return z;  Although this will compile and run, what kind of programming error would this be considered? 

Assume the arguments to foo() above are correct and the return statement in foo() is correct. Indicate which of the following statements are valid in our Reduced-C compiler and which will cause a compile error. [2pts each]

A) No Error in Reduced-C  
B) Compile Error in Reduced-C

```c
a = foo( /* Correct args */ );  ____
b = foo( /* Correct args */ );  ____
foo( /* Correct args */ ) = a;  ____
++foo( /* Correct args */ );  ____
```