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( 5 );
cout << x;
void beta()
    int x = 1; // local var declaration
cout << x;
set_x( 7 );
set_x( 2 );
alpha();
cout << x;
beta();
cout << x;
```

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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; int[2] y;};  
FOO* foo;
int x;
int *p = &x;
const int y = 5;
int[5] a;
   _____ f1()   _____ f2()   _____ &f2()   _____ *f1()
   _____ *(float *)p   _____ (float *)p   _____ *(float *)&y   _____ (float *)&y
   _____ &x   _____ x   _____ y   _____ x*a[1]   _____ a[2]   _____ *p
   _____ 17   _____ a   _____ p   _____ *&x&*p   _____ foo->x   _____ foo->y
```

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

```c
   ____ extern int x;   ____ int * foo( int x ) { return x; }   ____ float * bar ( float z );
   ____ int a[50];   ____ extern int fooz( int x, float y );   ____ struct fubar { int x; } sl;
```

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

```c
int i = 5;
float f = (int) i;
```
5. Fill in the names of the 5 main areas of the C/C++ Runtime Environment as laid out by most SPARC systems as described in class, and state what part(s) of a program are stored in each [10pts]:

<table>
<thead>
<tr>
<th>low memory</th>
<th>high memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```c
int a;
float b;

function : int & foo( int x, float & y ) { /* function body */ }
```

Use only variables a, b, and the expression b++ as possible arguments to the function foo().

Give an example function call to foo() that triggers an addressability error (and only this error).

```c
foo( _______________, _______________ );
```

Give an example function call to foo() that triggers an assignability error (and only this error).

```c
foo( _______________, _______________ );
```

Give an example function call to foo() that triggers an equivalence error (and only this error).

```c
foo( _______________, _______________ );
```

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