1. **Phase 0 Scoping Fix.** Fill in the blanks of the following Reduced-C program with correct types to test if your fix to the scoping bug present in the starterCode works correctly. If the scoping bug is fixed, this program should compile without error. If the bug is not fixed, this program should generate an assignment error at the line a = z;

```
_____ a;       // global a

function : int main() {

_____ a;    // local a

int z;

a = z;        // If fixed, this line will not cause an error!
// If not fixed, this line will cause an error!

return 0;
}
```

2. **Modifiable L-vals, Non-Modifiable L-vals, R-vals**

From the Reduced-C Spec (which follows closely the real C language standard), complete the following:

A) Addressable  C) Modifiable
B) Not Addressable  D) Not Modifiable

A Modifiable L-value is _____ and  _____.

A Non-Modifiable L-value is _____ and  _____.

An R-value is _____ and  _____.

Given the definitions below, indicate whether each expression is either a

A) Modifiable L-val  B) Non-Modifiable L-val  C) R-val

```
int[5] a;
const int y = 5;
int x;
int *p = &x;

_____ &x _____ a _____ y _____ x + y _____ a[2] _____ 42
_____ (float *)p _____ *(float *)p _____ (float *)&x _____ *(float *)&x

_____ x _____ *p _____ *&p _____ &*p _____ p
```
3. Type Inference. Consider the following Reduced-C definitions:

```c
const bool a = 5 _Op1_ 7;
const bool b = true _Op2_ false;
const int c = 5 _Op3_ 7;
```

For _Op1_, _Op2_, and _Op3_, list what operators are valid (i.e., cause no compile errors). The available operators are listed below. Some _Op#_ have more than one possible valid operator.

```c
==   &&   +   >=
```

_Op1_: ____________________________
_Op2_: ____________________________
_Op3_: ____________________________

4. Semantic Checks/Errors:
Given the following Reduced-C code fragment:
```c
int a;
bool b;

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

Using variables a, b, and the expression (a + 4) as possible arguments to the function foo() give an example function call to foo() that triggers an equivalence 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 addressability error (and only this error).

5. In Reduced-C (which again follows closely the real C standard) all typedefs use _____________ name equivalence. Struct operations (like =, ==, !=) use ______________ name equivalence.

6. Identify the following C constructs as either
   A) Definition        B) Pure Declaration
   ____ struct fofo;   ____ extern int * func1( int x, float y );
   ____ int x;        ____ int foo( int x ) { return x; }
   ____ extern float y;   ____ struct fubar { int x; } s1;