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 \( a = z; \)

\[
\text{________ } a; \quad // \text{global a}
\]

\[
\text{function : int main() {}
\]

\[
\text{________ } a; \quad // \text{local a}
\]

\[
\text{int z;}
\]

\[
\text{a = z;} \quad // \text{If fixed, this line will not cause an error!}
\]

\[
\text{// If not fixed, this line will cause an error!}
\]

\[
\text{return 0;}
\]

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  
B) Not Addressable
C) Modifiable  
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

\[
\text{int x;}
\]

\[
\text{const int y = 5;}
\]

\[
\text{int[5] a;}
\]

\[
\text{int *p = &x;}
\]

\[
\text{____ (float *)p  ____ *(float *)p  ____ (float *)&x  ____ *(float *)&x}
\]

\[
\text{____ &x  ____ a  ____ y  ____ x + y  ____ a[2]  ____ 42}
\]

\[
\text{____ x  ____ *p  ____ *&p  ____ &*p  ____ p}
\]
3. **Type Inference.** Consider the following Reduced-C program:

```c
const bool a = 5 __Op1__ 7;
const bool b = true __Op3__ false;
const int c = 5 __Op2__ 7;
float x;
bool z;

function : int main()
{
    if ( a ) { return 1; }
    
z = b;
x = c;
    return 0;
}
```

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.

- __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 + 2)` 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).

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