Login name ________________

Quiz 3
CSE 131
Winter 2011

Name ______________________

Signature __________________

Student ID _________________

1. Given the following C++ definitions (similar to Reduced-C)

```
struct S1 { int a; };
struct S2 { int a; };

void foo ( struct S2 &b ) { }

struct S1 a;
```

a call to `foo( a )` passing in `a` as the actual argument will cause a compiler error. <u>Why?</u>

Fix the function call `foo( a )` below to pass `a` to `foo()` without causing a C++ compiler error.

```
foo( ____________________________________________ a );
```

Using Reduced-C syntax, define a variable named `myPtr` that is pointer to an array of 5 ints. This will take two lines of code.

Show the memory layout of the following C struct definition taking into consideration the **SPARC** data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate struct member/field name. For example, if member/field name `p` takes 4 bytes, you will have 4 `p`'s in the appropriate memory locations. If the member/field is an array, use the name followed by the index number. For example, some number of `p[0]`s, `p[1]`s, `p[2]`s, etc. Place an `X` in any bytes of padding. Structs and unions are padded so the total size is evenly divisible by the most strict alignment requirement of its members.

```
struct foo {
   char    a;
   short   b;
   short  *c;
   short   d[3];
   double  e;
   short   f;
   char    g;
};

struct foo fubar;
```

<u>`lower memory`</u>

`fubar:`

<u>`higher memory`</u>

What is the `offsetof( struct foo, f )`? ________

What is the `offsetof( struct foo, a )`? ________

What is the `sizeof( struct foo )`? ________

If `struct foo` had been defined as `union foo` instead, what would be the `sizeof( union foo )`? ________

2. Given the following Reduced-C code and list of statements, indicate for each numbered statement the type of error that should be reported according to the Project I spec for this quarter (which is similar to C++ rules). Use the letters associated with the available errors in the box on the right, or choose E for No error.

```
int * [5] a;
const int b = 5;
bool c, d;
function : int foo(){ /* ... */ return 0; }
```

A) Operand to ++ is not a modifiable L-value.
B) Operand to & is not a modifiable L-value.
C) Left-hand operand is not assignable
(not a modifiable L-value).
D) Non-addressable argument to address-of operator.
E) No error.

```
____ 1) b = 3;

____ 2) &&a[0];

____ 3) a[b-2] = &b;

____ 4) (int)c = 4;

____ 5) &foo();

____ 6) *a[foo()] = b;

____ 7) a[2] = (int *) &c;

____ 8) ++b;

____ 9) (c = d) = true;

____ 10) c = d = true;

____ 11) (*a[0])++ = *a[1];

____ 12) *a[0]++ = *a[1];
```

State whether constant folding can be performed by the compiler according to this quarter's Reduced-C spec in the following Reduced-C statements (**Yes** or **No**)

```
function : void foo()
{
  const int a = 5;
  int b = 3;

  const int c = a + 10;       ______

  int[53 + c] d;              ______

  b = d[d[2] + c];            ______

  d[-2 + (a * b)] = c;        ______

  int e = d[a + c];           ______

  d[5 - 2 + c] = e;           ______

  b = d[e + a];               ______

  e = d[13 + b];              ______
}
```

Using only the following C variable declarations:

```
int a = 42;
int *b = &a;
float c = 4.20;
float *d = &c;
```

Give an example assignment stmt using a non-converting type cast (underlying bit pattern does not change).

Give an example assignment stmt using a converting type cast (underlying bit pattern changes).