1. Given the following Reduced-C code below, fill in the blanks of the compile error that should be reported according to this quarter's Project I spec. Use the letters associated with the words in the box below.

```c
typedef float F1;
typedef F1 F2;
typedef int I1;
typedef I1 I2;
I1 x;
I2 y;
F2 z;
```

```
x = z = y; // Compile error reported here. Assume this stmt is inside a function.
```

Value of type ____ not ____ to variable of type ____ .

2. 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.

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

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

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 )`? ________
Given the C array declaration

```c
double a[3];
```

Mark with an A the memory location(s) where we would find `a[1]`

If `a[0]` is allocated at memory location 5000, what value does `&a[2]` evaluate to? 

4. Given the following C definitions explain why the following are illegal expressions. Be specific.

```c
int array[5] = { 5, 4, 3, 2, 1 };
int *ptr = array;

**(a(&ptr[2]+1)) = *ptr++;           (short *)ptr = (short *)(array + 2);
```

5. Define an array of array of ints named `fubar` in Reduced-C such that

```
fubar[7][3]
```

is the last element in this data structure. You will need two lines of Reduced-C code to do this.

6. Why is the use of a traversal pointer to cycle through all the elements of a C/C++ multidimensional array almost always more efficient than using standard array indexing?