1. 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 {
    int     a;
    char    b;
    short   c[2];
    float   d;
    double *e;
    double  f;
    int     g;
};

struct foo fubar;
```

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

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

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 C array declaration

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

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

(a each box represents a byte in memory)

If `a[0]` is allocated at memory location 4000 (decimal), what value does `&a[2]` evaluate to? __________
3. Given the following Reduced-C definitions (similar to C++)

```c
structdef S1 { int a; }
structdef S2 { int a; }

void foo1( int a ) { }
void foo2( S1 &a ) { }

typedef int T1;
typedef T1 T2;
typedef S1 T3;
typedef T3 T4;

S1 a;
S2 b;
T1 c;
T2 d;
T3 e;
T4 f;
float g;
```

indicate whether each of the following statements will cause a compiler error or not. A) Error  
B) No Error

<table>
<thead>
<tr>
<th>Statement</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a = e;</code></td>
<td>____</td>
</tr>
<tr>
<td><code>foo1( d );</code></td>
<td>____</td>
</tr>
<tr>
<td><code>a = f;</code></td>
<td>____</td>
</tr>
<tr>
<td><code>foo1( g );</code></td>
<td>____</td>
</tr>
<tr>
<td><code>a = b;</code></td>
<td>____</td>
</tr>
<tr>
<td><code>foo2( a );</code></td>
<td>____</td>
</tr>
<tr>
<td><code>c = d;</code></td>
<td>____</td>
</tr>
<tr>
<td><code>foo2( f );</code></td>
<td>____</td>
</tr>
<tr>
<td><code>c = g;</code></td>
<td>____</td>
</tr>
<tr>
<td><code>foo2( e );</code></td>
<td>____</td>
</tr>
<tr>
<td><code>foo2( b );</code></td>
<td>____</td>
</tr>
</tbody>
</table>

4. Using Reduced-C syntax, define a pointer to an array of 5 floats named `foo` such that

```c
float x = 4.2;

(*foo)[4] = x;
x = (*foo)[4];
```

are valid expressions. This will take two lines of Reduced-C code.

5. In the following example

```c
typedef int INT;
INT x;
```

What type of STO do you need to create for `INT`? _______________________

What type of STO do you need to create for `x`? _______________________