Welcome back to section! We’re glad that you’re here :)  

1. Refer to the following file definitions. 

<table>
<thead>
<tr>
<th>File</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point.h</td>
<td>class Point { ... };</td>
</tr>
<tr>
<td>UsePoint.cc</td>
<td>#include &quot;Point.h&quot;</td>
</tr>
<tr>
<td></td>
<td>#include &quot;Thing.h&quot;</td>
</tr>
<tr>
<td></td>
<td>int main( ... ) { ... }</td>
</tr>
<tr>
<td>Point.cc</td>
<td>#include &quot;Point.h&quot;</td>
</tr>
<tr>
<td></td>
<td>// def of methods</td>
</tr>
<tr>
<td></td>
<td>Thing.h</td>
</tr>
<tr>
<td></td>
<td>struct Thing { ... };</td>
</tr>
<tr>
<td></td>
<td>// full struct def here</td>
</tr>
<tr>
<td></td>
<td>Alone.cc</td>
</tr>
<tr>
<td></td>
<td>int main( ... ) { ... }</td>
</tr>
</tbody>
</table>

a. Draw out Point’s DAG  
(The direction of the arrows is not important, but be consistent) 

Write the corresponding Makefile for Point.
References
References create aliases that we can bind to existing variables. References are not separate variables and cannot be reassigned after they are initialized. In C++, you define a reference using: `type& name = var`. The '!' is similar to the '*' in a pointer definition in that it modifies the type and the space can come before or after it.

Const
Const makes a variable *unchangeable* after initialization, and is enforced at compile time.

```cpp
const int x = 5; // Can't assign to x
const int* x_ptr = &x; // Can assign to x_ptr, but not *x_ptr
int* const y_ptr = &y; // Can assign to *y_ptr, but not y_ptr
const int* const z_ptr = &z; // Can't assign to *z_ptr or z_ptr
```

Class objects can be declared const too - a const class object can only call member functions that have been declared as const, which are not allowed to modify the object instance it is being called on.

Exercises:
2) Consider the following functions and variable declarations.
   a) Draw a memory diagram for the variables declared in `main`. It might be helpful to distinguish variables that are constant in your memory diagram.

```cpp
int main(int argc, char** argv) {  
    int x = 5;
    int& x_ref = x;
    int* x_ptr = &x;
    const int& ro_x_ref = x;
    const int* ro_ptr1 = &x;
    int* const ro_ptr2 = &x;
    // ...
}
```
b) When would you prefer `void Func(int &arg);` to `void Func(int *arg);`?
   Expand on this distinction for other types besides `int`.

c) If we have functions `void Foo(const int& arg);` and `void Bar(int& arg);`,
   what does the compiler think about the following lines of code:
   
   ```
   Bar(x_ref);
   Bar(ro_x_ref);
   Foo(x_ref);
   ```

d) How about this code?
   ```
   ro_ptr1 = (int*) 0xDEADBEEF;
   x_ptr = &ro_x_ref;
   ro_ptr2 = ro_ptr2 + 2;
   *ro_ptr1 = *ro_ptr1 + 1;
   ```