3. Consider the following function’s assembly code:

```assembly
00000000 <bar2>:
  0:  55 push %ebp
  1:  89 e5 mov %esp,%ebp
  3:  8b 45 08 mov 0x8(%ebp),%eax
  6:  8d 50 03 lea 0x3(%eax),%edx
  9:  85 c0 test %eax,%eax
 b:  0f 49 d0 cmovns %eax,%edx
 e:  c1 fa 02 sar $0x2,%edx
 11: b8 00 00 00 00 mov $0x0,%eax
 16: 85 d2 test %edx,%edx
 18: 7e 12 jle 2c <bar2+0x2c>
 1a: 8d 4a 03 lea 0x3(%edx),%ecx
 1d: 85 d2 test %edx,%edx
 1f: 0f 48 d1 cmovs %ecx,%edx
 22: c1 fa 02 sar $0x2,%edx
 25: 83 c0 01 add $0x1,%eax
 28: 85 d2 test %edx,%edx
 2a: 7f ee jg 1a <bar2+0x1a>
 2c: 5d pop %ebp
 2d: c3 ret
```

Fill in the corresponding C code:

```c
int bar(int x)
{
    int y = 0;
    int z = ________________;
    for( ; ___________ ; ___________ )
    {
        z = ________________;
    }
    return ________________;
}
```
4. Given the following assembly code:

```
xorl %eax, %eax
leal 16(%ecx), %ebx
.L59:
    leal (%eax, %eax, 4), %edx
movl (%ecx), %eax
addl $4, %ecx
leal (%eax, %edx, 2), %eax
cmp  %ebx, %ecx
jle .L59
```

Put comments on each assembly statement as it pertains to the C code then give equivalent C code. Here is some additional information that will help:

- Register assignments:
  - %ecx is z
  - %eax is zi
  - %ebx is zend

- Computations:
  - $10*zi + *z$ is implemented as $*z + 2*(zi+4*zi)$
  - $Z++$ increments by 4

5. Using the template below (allowing a maximum of 40 bytes, indicate the allocation of data for struct my_struct. Mark off and label the areas for each individual element (arrays may be labeled as a single element) where each cell in the template is 1 byte. Shade the boxes used for padding i.e. the ones that are allocated, but not used; and be sure to clearly indicate the end of the structure. Use the letter of the variable to designate the space filled for each byte.

```
struct my_struct  {
    short b;
    int x;
    short s;
    long z;
    char c[5]
    long long a;
    char q;        }
```

<table>
<thead>
<tr>
<th>b</th>
<th>x</th>
<th>s</th>
<th>z</th>
<th>c</th>
<th>a</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. Below is the C code and assembly code for a simple function. Draw a detailed stack diagram for this function, starting with a function that calls this function and continuing for 2 recursive calls of this function i.e. at least two stack frames that belong to this function. Be sure to label everything you can so your solution is understandable.

```c
int doSomething(int a, int b, int c)
{
    int d;
    if (a == 0) { return 1; }
    d = a/2;
    c = doSomething(d,a,c);
    return c;
}
```

```assembly
000000af <doSomething>:       int doSomething(int a, int b, int c){
af:   push   %ebp
b0:   mov    %esp,%ebp
b2:   sub    $0xc,%esp
b5:   mov    0x8(%ebp),%ecx
b8:   mov    $0x1,%eax
bd:   test   %ecx,%ecx
bf:   je     de <doSomething+0x2f>
c1:   mov    %ecx,%edx
c3:   shr    $0x1f,%edx
c6:   lea    (%ecx,%edx,1),%edx
c9:   sar    %edx
cb:   mov    0x10(%ebp),%eax
ce:   mov    %eax,0x8(%esp)
d2:   mov    %ecx,0x4(%esp)
d6:   mov    %edx,(%esp)
d9:   call   da <doSomething+0x2b>
de:   leave
df:   ret
```

7. Give the IA32 instruction format for each of the following assembly statements.
   A. push %ebp
   B. sub $0x24, %esp
   C. add $0xffffffff8, %esp
   D. lea 0xf8(%ebp), %ebx
   E. push $0x804857b
   F. test %eax, %eax
   G. pop %ebp
   H. ret