When this function is called, identify which area of the C Runtime Environment each of the following will be allocated and its scope or visibility.

<table>
<thead>
<tr>
<th>Area of Runtime Env.</th>
<th>Scope/Visibility (Global/File/Block)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
</tr>
<tr>
<td>fubar</td>
<td></td>
</tr>
</tbody>
</table>

Where c is pointing ______

If the function above is called 17 times, indicate how many times will d be initialized to 42? ______

Convert 133.375<sub>10</sub> to binary fixed-point and single precision IEEE floating-point representation (expressed in hexadecimal).

- Binary fixed-point: __________________________ x 2<sup>0</sup>
- IEEE floating-point: __________________________ (hexadecimal)

Convert 0xC2D000 (single precision IEEE floating-point representation) to fixed-point decimal.

- Fixed-point decimal: __________________________ (decimal / no exponential notation)

Using the Rt-Lt Rule, declare bar to be an array of 9 elements where each element is a pointer to a function that takes a pointer to a int and returns a pointer to a struct foo.
Given the following program, order the line numbers so that the addresses that are printed are ordered from smallest to largest if compiled and run on our ARM pi-cluster architecture. These lines print out the hex address of the different parts of the program (not the values assigned) with the printf() format specifier %p (pointer). Basically, where do the different parts of a C program live in the run time environment ordered from smaller addresses to larger addresses?

```c
#include <stdio.h>
#include <stdlib.h>

int a = 42;
void foo( int b ) {
    int c = 420;
    /* 1 */ (void) printf( "1:a --> %p\n", &a );
    /* 2 */ (void) printf( "2:b --> %p\n", &b );
    /* 3 */ (void) printf( "3:c --> %p\n", &c );
}

int main( int argc, char *argv[] ) {
    static int d;
    int e = 420;
    int f;
    foo(e); /* Call function foo() */
    /* 4 */ (void) printf( "4:d --> %p\n", &d );
    /* 5 */ (void) printf( "5:e --> %p\n", &e );
    /* 6 */ (void) printf( "6:malloc --> %p\n", malloc(e) );
    /* 7 */ (void) printf( "7:argc --> %p\n", &argc );
    /* 8 */ (void) printf( "8:f --> %p\n", &f );
    /* 9 */ (void) printf( "9:argv --> %p\n", &argv );
    /*10 */ (void) printf( "10:foo() --> %p\n", foo );
    return 0;
}
```

This line number would print the smallest address

This line number would print the largest address

Use the numbers in the box to answer the following (numbers may be used more than once or not at all)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>pure declaration</td>
<td>linkage editor</td>
<td>extern keyword</td>
<td>default linking</td>
<td>reports multiply-defined symbols</td>
<td>translates assembly code</td>
<td>resolves undefined symbols</td>
<td>ensures bss segment is set up</td>
<td>allocated in text, data, or bss</td>
<td>not listed in a .global directive</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>loader</td>
<td>assembler</td>
<td>compiler</td>
<td>static</td>
<td>pre-processor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>