Write the equivalent ARM assembly language instructions to make the following function call. Assume the struct is the only local variable. Allocate and lay out the struct properly on the stack as we did in class. Use magic numbers in your code to make it easier for you to code and for us to grade your code. Just write the ARM assembly code to allocate stack space for the local variable fb and pass the specified values in to the function call and save the return value in the specified value.

```c
/* Function prototype for foo() */
char foo( char, short, int );

int main() {
    /* Only local variable */
    /* Allocate stack space for var fb */
    struct fubar {
        int     a[2];
        char    b;
        short   c[6];
        char    d[2];
    } fb;

    /* Write the equivalent ARM assembly code for just this function call */
    /* saving the return value appropriately. */
    fb.d[0] = foo( fb.b, fb.c[0], fb.a[1] );
    /* Other code */
}

main:
    push {fp, lr}
    add fp, sp, 4
    # Allocate stack space for local var fb

    # Pass the specified args appropriately
    # and call the function saving the return value as specified.

    # Other code - Assume other code from here
    # to end are correct.
```
What is the output of the following program? (Hint: Draw stack frames!)

```c
int mystery( int param ) {
    int local = 8;
    if ( local >= param ) {
        local = param + local;
        /* Output the value of local followed by a newline */
        printf( "%d\n", local );
        param = local + mystery( local - 6 );
        /* Output the value of param followed by a newline */
        printf( "%d\n", param );
    } else {
        printf( "Stop\n" );
    }
    return local;
}
```

What gets printed with the function call `mystery( 4 );`? (Hint: Draw stack frames!)

```c
int main() {
    int a = 6;
    int b = 12;
    swap1( a, b );
    printf( "\n", a );
    printf( "\n", b );
    a = 7;
    b = 4;
    swap2( &a, b );
    printf( "\n", a );
    printf( "\n", b );
    a = 3;
    b = 8;
    swap3( &a, &b );
    printf( "\n", a );
    printf( "\n", b );
    return 0;
}
```

Using the Rt-Lt Rule, declare `foo` to be a function that takes a pointer to a float and returns a pointer to an array of 23 elements where each element is a pointer to a struct bar.

```c
void swap1( int a, int b ){
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}

void swap2( int *a, int b ){
    int tmp;
    tmp = *a;
    *a = b;
    b = tmp;
}

void swap3( int *a, int *b ){
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}
```