Programming #1 -- simple C aliasing problem

```
/*
Name:               Tony Maldonado
Date:               August 31, 2020

Input:              None.

Output:             A sentence which displays your name, the addresses of the
                    array and the pointer to the array.

Preconditions:      Your computer has to be running on Little Endian for it
                    to output correctly.

Postconditions:     None.

*/

#include <stdio.h>

// Global declaration of A for expirement of question 3.c.
//int A[100];

int main (){

        // Static declaration of A for expirement of quesiton 3.c.
        // static int A[100];

        int A[100];
        char *S;

        A[0] = 84 + (111 * 256) + (110 * 256 * 256) + (121 * 256 * 256 * 256);
        A[1] = 32 + (77 * 256) + (97 * 256 * 256) + (108 * 256 * 256 * 256);
        A[2] = 100 + (111 * 256) + (110 * 256 * 256) + (97 * 256 * 256 * 256);
        A[3] = 100 + (111 * 256) + 0;
        //A[4] = 0;

        S = (char *) A;


        printf("My name is %s\n", S);

        printf("Array A      is located at %20u \n", A);
        printf("Pointer S    is located at %20u \n", S);
```

```
}
```

/*

2. (Screenshot of program running below).

3.a. The array is allocated in the stack (dynamic) segment of memory (screenshot below).

3.b. The pointer is also allocated in the stack (dynamic) segment of memory, in the same address as as the array, since it points to it (screenshot below).

3.c. By making it global or static (screenshot below).

3.d. My computer runs on Little Endian because it is placing the least significant byte at the lowest address. If it was big endian, it would print "My name is ynoT odanodlaM".

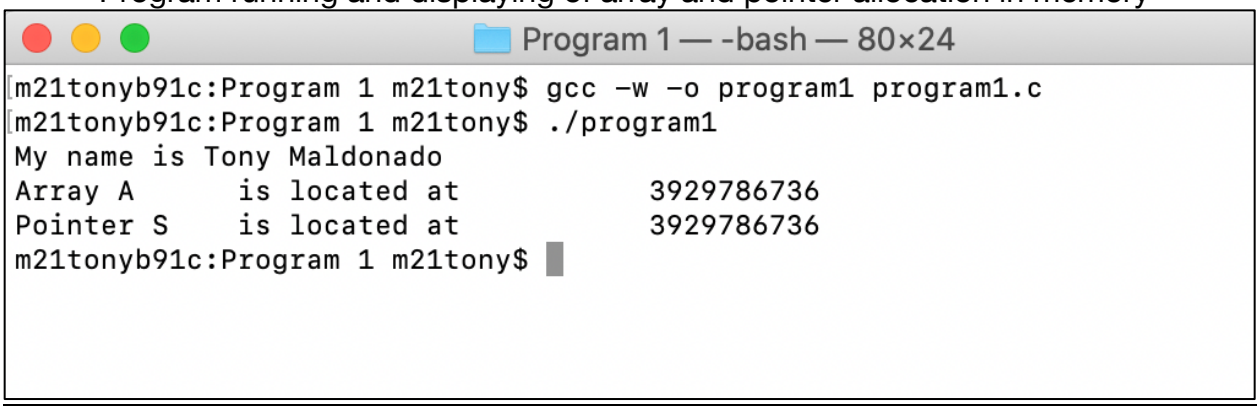
3.e. The difference is that in big endian it places the most significant byte, so, the 2^24...2^31 bits in the lowest address. And in little endian, it's the opposite. So, it places the 2^24...2^31 bits in the highest address. In this program's case, the '84' in A[0] would be the least significant byte which gets placed at the lowest address, and it places the highest byte in the highest address.

4. No, we don't need to fill the entire last integer with '0'. In A[3], I only filled two bytes then added '0' which filled in the other 2 bytes with '0', or 'null' and it worked correctly. Basically, you can do either.

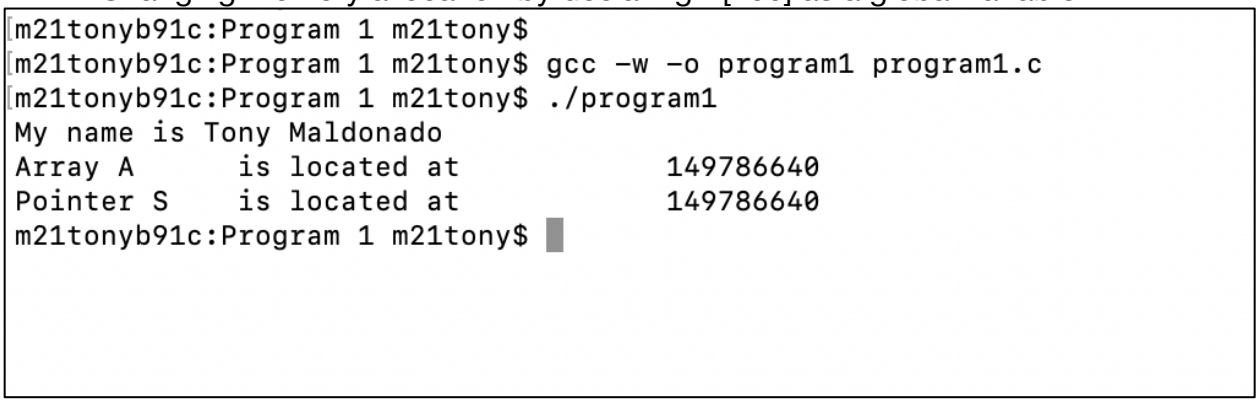
*/

Screenshots:

- Program running and displaying of array and pointer allocation in memory



- Changing memory allocation by declaring A[100] as a global variable



- Changing memory allocation by declaring A[100] as a static variable