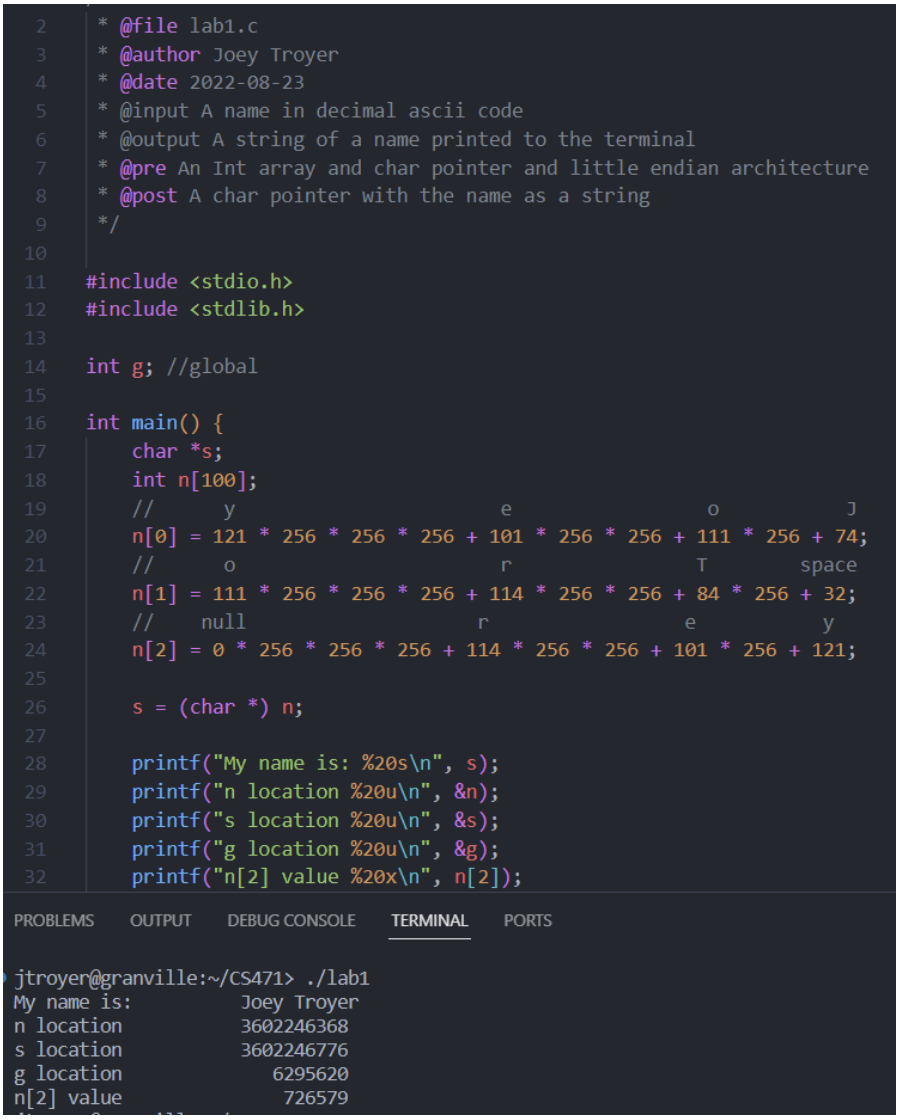
2.



```
 * @file lab1.c
 * @author Joey Troyer
 * @date 2022-08-23
 * @input A name in decimal ascii code
 * @output A string of a name printed to the terminal
 * @pre An Int array and char pointer and little endian architecture
 * @post A char pointer with the name as a string
 */

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

int g; //global

int main() {
    char *s;
    int n[100];
    //      y                       e                 o           J
    n[0] = 121 * 256 * 256 * 256 + 101 * 256 * 256 + 111 * 256 + 74;
    //      o                       r               T        space
    n[1] = 111 * 256 * 256 * 256 + 114 * 256 * 256 + 84 * 256 + 32;
    //    null                    r                 e          y
    n[2] = 0 * 256 * 256 * 256 + 114 * 256 * 256 + 101 * 256 + 121;

    s = (char *) n;

    printf("My name is: %20s\n", s);
    printf("n location %20u\n", &n);
    printf("s location %20u\n", &s);
    printf("g location %20u\n", &g);
    printf("n[2] value %20x\n", n[2]);
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

jtroyer@granville:~/CS471> ./lab1
My name is:          Joey Troyer
n location           3602246368
s location           3602246776
g location              6295620
n[2] value               726579
```

a. The array is in the stack. We can look at the address of the global variable which is stored in the hemp, and the address of the array. The array is such a bigger number than the global, and since the stack is at the top, it is most likely stored in the stack.

b. The pointer is on the stack. We can prove this because it is 416 bytes away from the array. It is an int array of 100, and each int is 4 bytes, so this would make sense. It is also very far away from the global variable.

c. We change the location of where an array is stored by declaring a global array outside of the main.

```
14    int g; //global
15    int x[100]; //global array
16    int main() {
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    T

jtroyer@granville:~/CS471> ./lab1
My name is:          Joey Troyer
n location           702950656
x location             6295648
```

d. Little-endian

e. Big endian stores value big values first and then smaller values so that you would read it left to right. Little-endian stores the smaller values first, then the bigger values, so you would read it right to left. Little-endian can seem counter-intuitive, but it could allow for faster computation times. This is because in big-endian, with each increase in magnitude, each byte would have to shift to the right, while in little-endian, every increase in magnitude adds a digit to left, and every smaller digit stays in the same place.

https://www.techtarget.com/searchnetworking/definition/big-endian-and-little-endian#:~:text=Big%2Dendian%20is%20an%20order,the%20sequence)%20is%20stored%20first.

f. We just need to change the first byte to a zero for the string to stop. If I change the place of the y to a 0, the string only prints up to the place of y and disregards the characters after.

```
23    //    null                    r                  e          y
24    n[2] = 121 * 256 * 256 * 256 + 114 * 256 * 256 + 101 * 256 + 0;
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

jtroyer@granville:~/CS471> ./lab1
My name is:             Joey Tro
n location           1287315792
s location           1287316200
g location              6295620
n[2] value             79726500
```