Xiana Lara
Programming #1
September 1, 2020

Programming #1 -- simple C aliasing problem

Screenshot of my program running:

```c
// Xiana Lara
// September 1st 2020
// Programming #1
// input: N/A
// output: print statements containing
// my name and memory allocations

#include <stdio.h>

int main(){

    int arr[3];
    char *S;

    // Name: Xiana Lara
    // Xian
    arr[0] = (88 + (105 * 256) + (97 * 256 * 256) + (110 * 256 * 256 * 256));
    // a_La
    arr[1] = (97 + (32 * 256) + (76 * 256 * 256) + (97 * 256 * 256* 256));
    // ra and ending final integer with a 0
    arr[2] = (114 + (97 * 256) + 0);


    S = (char*)arr;

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

    // Question a
    printf("\nThe memory segment where the array is allocated is: %u\n", &arr);

    // Question b
    printf("The memory segment where the pointer to the array is allocated is: %u\n", &S);

} // of main
```

```
MINGW64:/c/Users/xiana/OneDrive/Documents/school/PLStruc
xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc
$ gcc prog1xl.c -o prog1xl

xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc
$ ./prog1xl
My name is Xiana Lara

The memory segment where the array is allocated is: 6422212
The memory segment where the pointer to the array is allocated is: 6422208

xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc
$
```

a) In what memory segment is the array allocated? Give proof that your answer is correct

The array is stored in the stack with a memory segment allocation at 6422212. This can be seen in the screenshot below.

```
xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc
$ ./prog1xl
My name is Xiana Lara

The memory segment where the array is allocated is: 6422212
The memory segment where the pointer to the array is allocated is: 6422208
```

b) In what memory segment is the pointer to the array allocated? Give proof that your answer is correct

The pointer to the array is stored in the stack with a memory segment allocation at 6422208. This can be seen in the screenshot above.

c) How can you make your array be in another segment? Show how you did this and show proof

If the array is static, then it will be allocated to the data segment instead of the stack. You can see this in the screenshot below.

Xiana Lara
Programming #1
September 1, 2020

```c
// Xiana Lara
// September 1st 2020
// Programming #1
// input: N/A
// output: memory addresses

#include <stdio.h>

int main(){

    int arr[3];
    char *s;

    printf("\nArray address: %u\n", &arr);
    printf("Pointer address: %u\n\n", &s);

    static int statArr[3];
    printf("\nStatic array address: %u\n", statArr);

} // of main
```

```
MINGW64:/c/Users/xiana/OneDrive/Documents/school/PLStruc/prg1

xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc/prg1
$ ./test1

Array address: 6422212
Pointer address: 6422208


Static array address: 4218932

xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc/prg1
$
```

d) What endianness was the computer you ran your problem on?

My laptop is little endian and could be seen from the test below. It shows that the least significant is the first character, which is little endian.

```
MINGW64:/c/Users/xiana/OneDrive/Documents/school/PLStruc/prg1

xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc/prg1
$ perl -e 'use Config; print $Config{byteorder}'
12345678
xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc/prg1
$
```

Xiana Lara
Programming #1
September 1, 2020

e) Why is there a difference between little and big endian? Which one is better?

The difference is the way in which the bits are ordered. Little endian being the least significant first and big endian being most significant first. Neither is significantly better than the other, the most important thing is consistency.

Do you we need to fill the entire last interger with '0', or can we just fill in the last byte with '0'. Show an experiment that shows this (make sure you pay attention to endianness and ensure that your other bytes are NOT 0 when doing the experiment.

The results are the same either way, so it is not necessary to allocate the entire last integer. You can see this in the screenshot below.

```c
// Xiana Lara
// September 1st 2020
// Programming #1
// input: N/A
// output: print statements containing
// my name and memory allocations
#include <stdio.h>

int main(){

    int arr[4];
    char *S;

    // Name: Xiana Lara
    // Xian
    arr[0] = (88 + (105 * 256) + (97 * 256 * 256) + (110 * 256 * 256 * 256));
    // a_La
    arr[1] = (97 + (32 * 256) + (76 * 256 * 256) + (97 * 256 * 256* 256));
    // ra and ending final integer with a 0
    arr[2] = (114 + (97 * 256) + 0);

    S = (char*)arr;
    printf("My name is %s \n", S);

    // Name: Xiana Lara
    // Xian
    arr[0] = (88 + (105 * 256) + (97 * 256 * 256) + (110 * 256 * 256 * 256));
    // a_La
    arr[1] = (97 + (32 * 256) + (76 * 256 * 256) + (97 * 256 * 256* 256));
    // ra and ending final integer with a 0
    arr[2] = (114 + (97 * 256));
    arr[3] = 0;

    S = (char*)arr;
    printf("My name is %s \n", S);

} // of main
```

```
MINGW64:/c/Users/xiana/OneDrive/Documents/school/PLStruc/prg1
$ gcc prgtst2.c -o test2

xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc/prg1
$ ./test2
My name is Xiana Lara
My name is Xiana Lara

xiana@DESKTOP-BC9SDU7 MINGW64 ~/OneDrive/Documents/school/PLStruc/prg1
$
```