Name: Pratyay Kumar
Aggie Id: 800811773

Answers for Programming#1 -- Simple C aliasing problem

1) The code is properly commented
2) Screenshot of my Program and Experiment is with name: Program: Programming#1.c - running.

3)

```
My name is Pratyay Kumar.
Array A is located at:              1871819416: Stack Address
The pointer to A is located at:     1871819400: Stack Address
Address of Test_Variable:           1871819384: Stack Address
Value of Test_Variable is at:        937453424: Heap Address
Address of Global Variable:            5160960: Data Segment Address
```

a) The Array A [100] is Allocated in: STACK.
b) The Pointer to A is in: STACK.
c) The array A [] is currently in STACK, to make the Array in another segment we can dynamically allocate the array. Here I have made a pointer array named test_variable, whose values are stored in Heap Address.

d) Little endianness.
e) The endianness name has come from swift Gulliver's travel story war, because of whether the boiled egg should be cracked open at the big end or at the small end.

   Endianness in computer science is about how bytes are stored in memory and not how we deal with the memory. In a machine which has memory location greater than a byte, those bytes must be stored in memory in particular order. So, we can put the most significant byte first (big endian) or the least significant byte first (little endian) at the byte with the lowest address, this is called endianness. Endianness is important because not knowing how data is stored would lead to communicating different values.

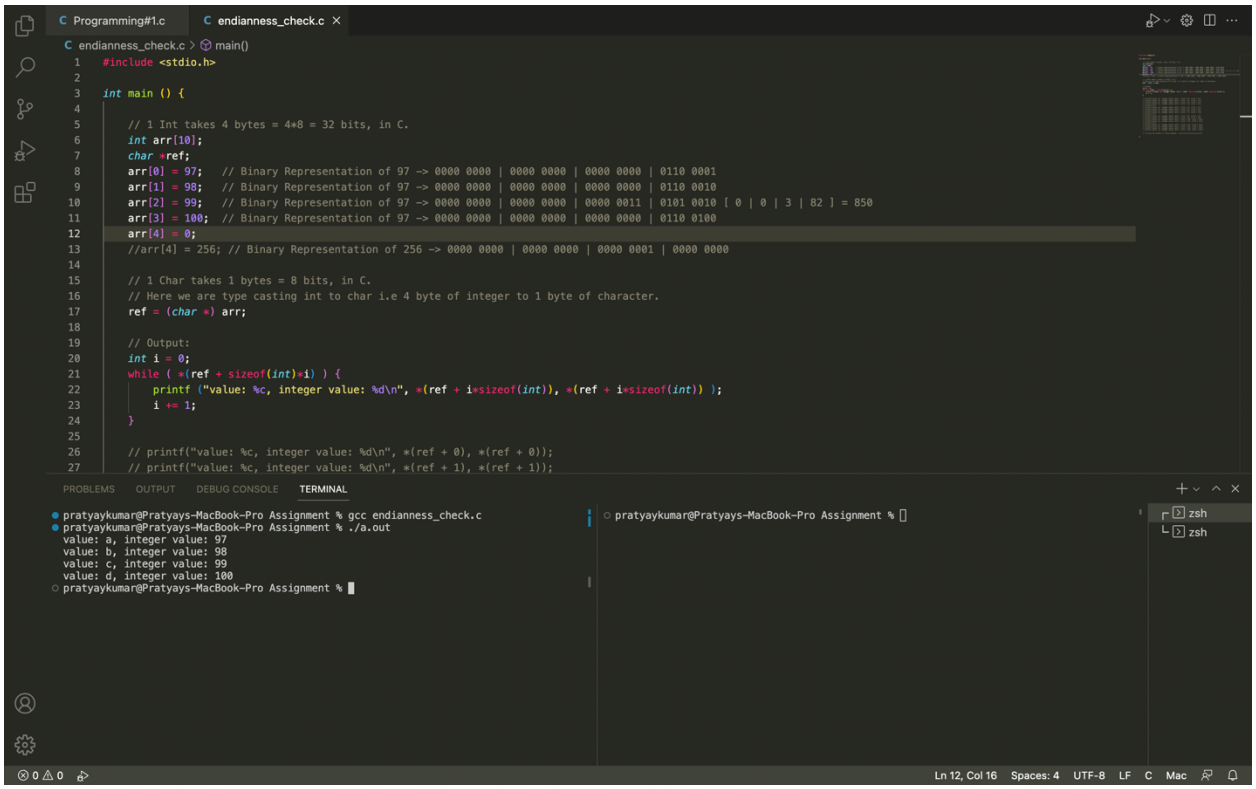
   We cannot really say big endian is better or little endian is better. Both has their own advantages. Big endian is commonly used for networking purpose. If we want to transfer data between two different systems over a network or on a file, or construct then when receiving, we need to know which way that data is been represented.

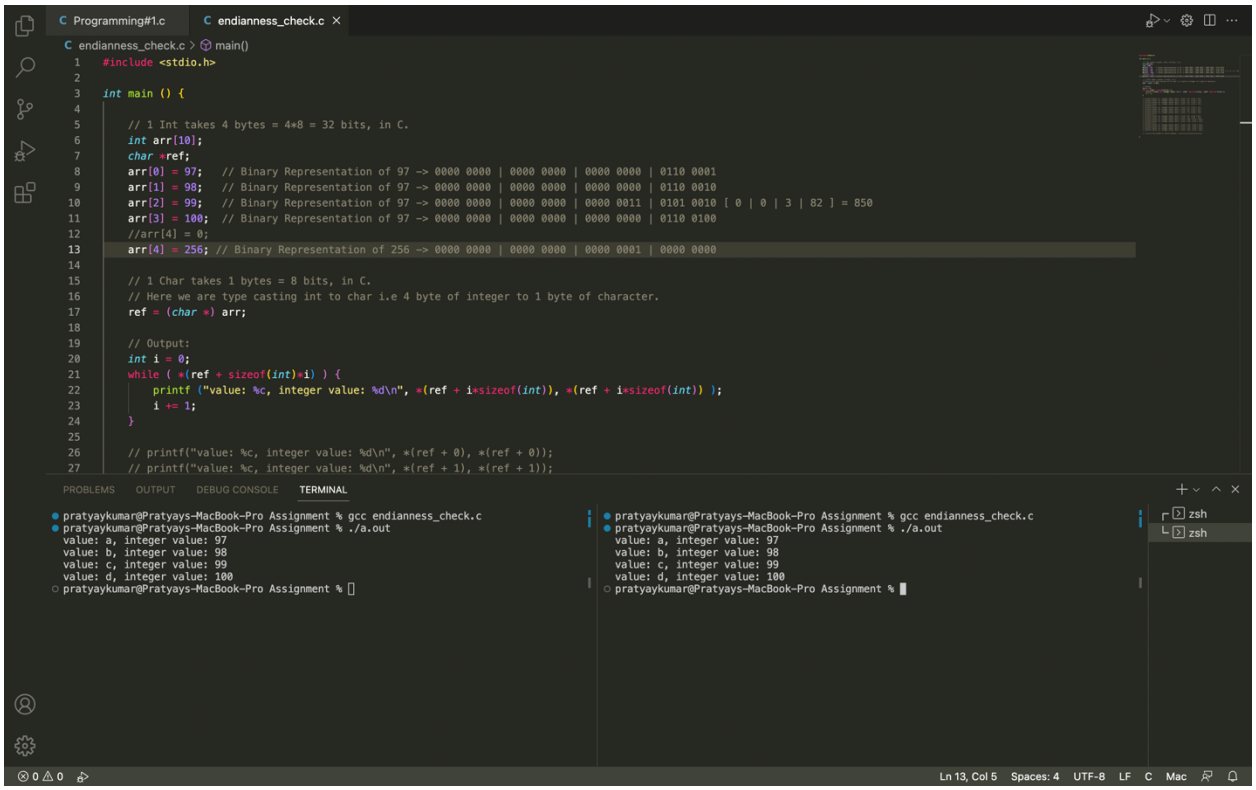
   Source: https://en.wikipedia.org/wiki/Endianness
   Source: https://www.section.io/engineering-education/what-is-little-endian-and-big-endian/

4) No, we don't need to fill the entire last integer with '0'. We can fill the last byte of with '0' to terminate while () loop.

   In the experiment I have taken arr [4] = 256, whose binary representation is 0000 0000 0000 0000 0000 0001 0000 0000. The code works fine, because my PC is little endian, it takes the last byte first in the while loop, and since the last byte is zero, the while loops become false, and hence the code works fine.

   Below I have attached the screenshot of the experiment, The first one with arr [4] = 0, and the second one with arr [4] = 256. The code is present in endianness_check.c file.


Programming#1.c
endianness_check.c
endianness_check.c > main()
#include <stdio.h>
int main () {
// 1 Int takes 4 bytes = 4*8 = 32 bits, in C.
int arr[10];
char *ref;
arr[0] = 97; // Binary Representation of 97 -> 0000 0000 | 0000 0000 | 0000 0000 | 0110 0001
arr[1] = 98; // Binary Representation of 97 -> 0000 0000 | 0000 0000 | 0000 0000 | 0110 0010
arr[2] = 99; // Binary Representation of 97 -> 0000 0000 | 0000 0000 | 0000 0011 | 0101 0010 [0 | 0 | 3 | 82] = 850
arr[3] = 100; // Binary Representation of 97 -> 0000 0000 | 0000 0000 | 0000 0000 | 0110 0100
arr[4] = 0;
//arr[4] = 256; // Binary Representation of 256 -> 0000 0000 | 0000 0000 | 0000 0001 | 0000 0000
// 1 Char takes 1 bytes = 8 bits, in C.
// Here we are type casting int to char i.e 4 byte of integer to 1 byte of character.
ref = (char *) arr;
// Output:
int i = 0;
while (*(ref + sizeof(int)*i)) {
printf ("value: %c, integer value: %d\n", *(ref + i*sizeof(int)), *(ref + i*sizeof(int)));
i += 1;
}
// printf("value: %c, integer value: %d\n", *(ref + 0), *(ref + 0));
// printf("value: %c, integer value: %d\n", *(ref + 1), *(ref + 1));
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
pratyaykumar@Pratyays-MacBook-Pro Assignment % gcc endianness_check.c
pratyaykumar@Pratyays-MacBook-Pro Assignment % ./a.out
value: a, integer value: 97
value: b, integer value: 98
value: c, integer value: 99
value: d, integer value: 100
pratyaykumar@Pratyays-MacBook-Pro Assignment %
pratyaykumar@Pratyays-MacBook-Pro Assignment %
zsh
zsh
0 0
Ln 12, Col 16 Spaces: 4 UTF-8 LF C Mac


Programming#1.c
endianness_check.c
endianness_check.c > main()
#include <stdio.h>
int main () {
// 1 Int takes 4 bytes = 4*8 = 32 bits, in C.
int arr[10];
char *ref;
arr[0] = 97; // Binary Representation of 97 -> 0000 0000 | 0000 0000 | 0000 0000 | 0110 0001
arr[1] = 98; // Binary Representation of 97 -> 0000 0000 | 0000 0000 | 0000 0000 | 0110 0010
arr[2] = 99; // Binary Representation of 97 -> 0000 0000 | 0000 0000 | 0000 0011 | 0101 0010 [0 | 0 | 3 | 82] = 850
arr[3] = 100; // Binary Representation of 97 -> 0000 0000 | 0000 0000 | 0000 0000 | 0110 0100
//arr[4] = 0;
arr[4] = 256; // Binary Representation of 256 -> 0000 0000 | 0000 0000 | 0000 0001 | 0000 0000
// 1 Char takes 1 bytes = 8 bits, in C.
// Here we are type casting int to char i.e 4 byte of integer to 1 byte of character.
ref = (char *) arr;
// Output:
int i = 0;
while (*(ref + sizeof(int)*i)) {
printf ("value: %c, integer value: %d\n", *(ref + i*sizeof(int)), *(ref + i*sizeof(int)));
i += 1;
}
// printf("value: %c, integer value: %d\n", *(ref + 0), *(ref + 0));
// printf("value: %c, integer value: %d\n", *(ref + 1), *(ref + 1));
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
pratyaykumar@Pratyays-MacBook-Pro Assignment % gcc endianness_check.c
pratyaykumar@Pratyays-MacBook-Pro Assignment % ./a.out
value: a, integer value: 97
value: b, integer value: 98
value: c, integer value: 99
value: d, integer value: 100
pratyaykumar@Pratyays-MacBook-Pro Assignment %
pratyaykumar@Pratyays-MacBook-Pro Assignment % gcc endianness_check.c
pratyaykumar@Pratyays-MacBook-Pro Assignment % ./a.out
value: a, integer value: 97
value: b, integer value: 98
value: c, integer value: 99
value: d, integer value: 100
pratyaykumar@Pratyays-MacBook-Pro Assignment %
zsh
zsh
0 0
Ln 13, Col 5 Spaces: 4 UTF-8 LF C Mac