The output of the code:

```
1: a = 0x7ffe6747c0f0, b = 0x55de01d1c260, c = 0xf0b5ff
2: a[0] = 200, a[1] = 101, a[2] = 102, a[3] = 103
3: a[0] = 200, a[1] = 300, a[2] = 301, a[3] = 302
4: a[0] = 200, a[1] = 400, a[2] = 301, a[3] = 302
5: a[0] = 200, a[1] = 128144, a[2] = 256, a[3] = 302
6: a = 0x7ffe6747c0f0, b = 0x7ffe6747c0f4, c = 0x7ffe6747c0f1
```

Justifications and Memory:

1: Justification: The printf statement prints the memory location that each pointer is pointing to through the “%p” identifier. Pointer (a) is pointing to a memory location in the stack, (b) is pointing to memory location in the heap, the value of (c) is garbage.

a = 0x7ffe6747c0f0	b = 0x55de01d1c260	c = 0xf0b5ff

Memory location	Value
a[0]	garbage
a[1]	garbage
a[2]	garbage
a[3]	garbage

Memory location	Value
b[0]	garbage
b[1]	garbage
b[2]	garbage
b[3]	garbage

2: Justification: Before the for loop, the instruction (c = a) makes (c) and (a) point to the same memory location. The for loop initializes the elements of a[i] with 100 + i, so a[0] = 100, a[1] = 101, a[2] = 102, a[3] = 103. Then c[0] which is the same as a[0] is set to 200.

a = 0x7ffe6747c0f0	b = 0x55de01d1c260	c = 0x7ffe6747c0f0

Memory location	Value
a[0]	200
a[1]	101
a[2]	102
a[3]	103

Memory location	Value
b[0]	garbage
b[1]	garbage
b[2]	garbage
b[3]	garbage

3: Justification: c and a are pointing to the same memory location, so c[0] = 200, c[1] which is a[1] is set to 300.

*(c + 2) = c[2] = a[2] = 301. 3[c] = c[3] = a[3] = 302.

a = 0x7ffe6747c0f0	b = 0x55de01d1c260	c = 0x7ffe6747c0f0

Memory location	Value
a[0]	200
a[1]	300
a[2]	301
a[3]	302

Memory location	Value
b[0]	garbage
b[1]	garbage
b[2]	garbage
b[3]	garbage

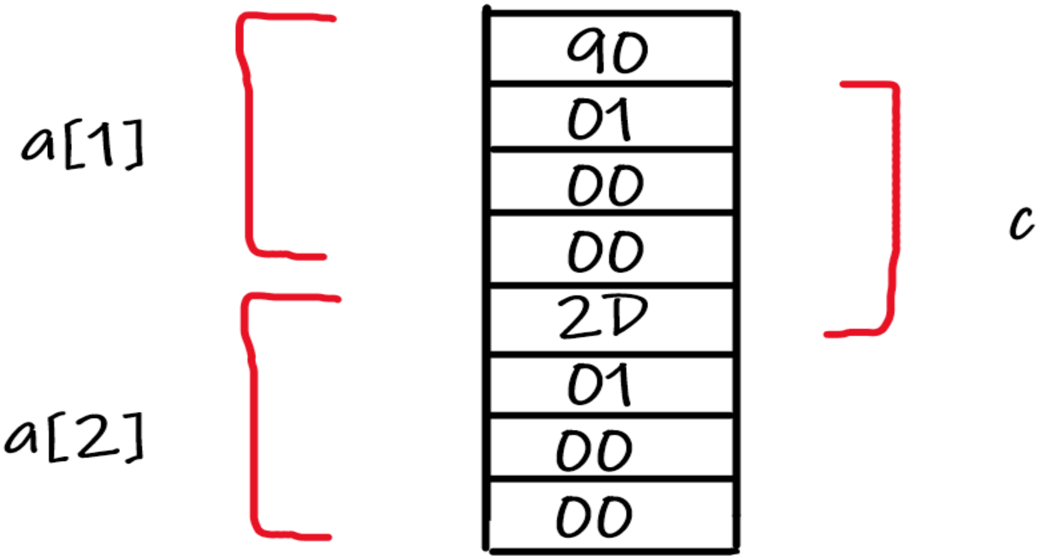
4: Justification: The expression (c = c + 1) makes c points to a[1] (it advances the pointer by 4 bytes because the size of int is 4 bytes) which means that (c[0] = a[1]). The statement (*c = 400) sets a[1] to 400.

a = 0x7ffe6747c0f0	b = 0x55de01d1c260	c = 0x7ffe6747c0f4

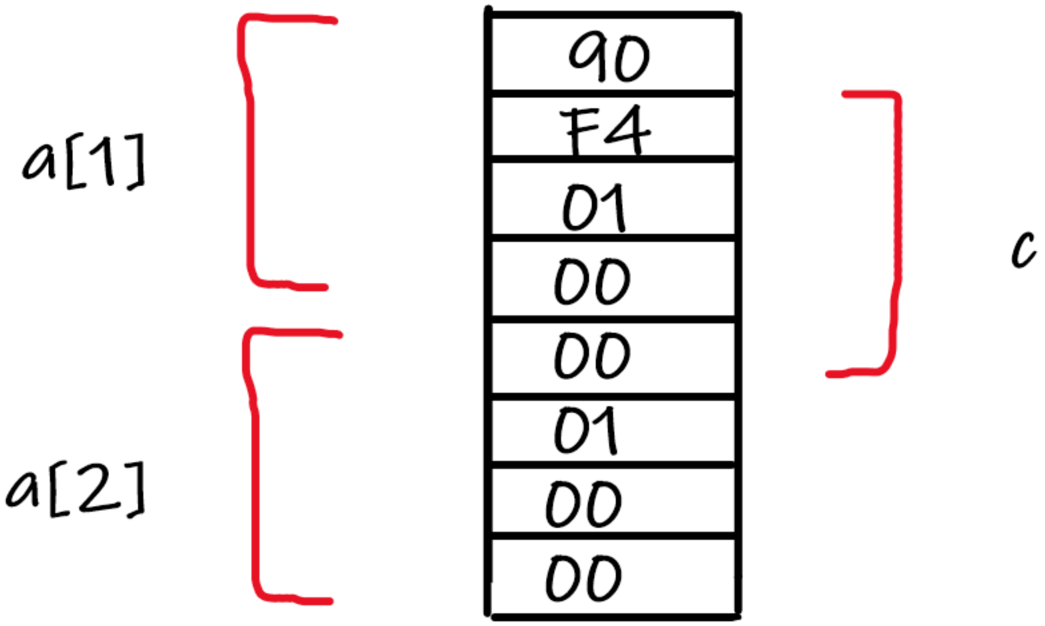
Memory location	Value
a[0]	200
a[1]	400
a[2]	301
a[3]	302

Memory location	Value
b[0]	garbage
b[1]	garbage
b[2]	garbage
b[3]	garbage

5: Justification: The pointer (c) is casted to char* then it is advanced by 1 (which means it is advanced by only one byte because the size of char is one byte) so (c) now is pointing to the second byte of a[1], and then casting it to be int* which means that it is currently pointing to a four bytes int (second byte of a[1], third byte of a[1], fourth byte of a[1], first byte of a[2]). The initial memory diagram is shown in the following figure.



After executing the statement (*c = 500 = 0x000001F4), The memory diagram is shown in the following figure (note that the memory is little endian).



a = 0x7ffe6747c0f0	b = 0x55de01d1c260	c = 0x7ffe6747c0f5

Memory location	Value
a[0]	200
a[1]	128144
a[2]	256
a[3]	302

Memory location	Value
b[0]	garbage
b[1]	garbage
b[2]	garbage
b[3]	garbage

6: Justification: The address which (a) is pointing to is not changed so it is the same as the initial address. The statement “b = (int *) a + 1” causes pointer b to point four bytes after the pointer (a), so (b) is pointing to a[1]. The statement “c = (int *) ((char *) a + 1)” causes pointer (c) to point one byte after the pointer (a).

c[0] = 0x90000000 = -1879048192 (if it is signed int) = 2415919104 (if it is unsigned int).

a = 0x7ffe6747c0f0	b = 0x7ffe6747c0f4	c = 0x7ffe6747c0f1

Memory location	Value
a[0]	200
a[1]	128144
a[2]	256
a[3]	302

Memory location	Value
b[0]	128144
b[1]	256
b[2]	302
b[3]	garbage