printf("1: a = %p, b = %p, c = %p, i = %p\n", a, &b, &c, &i);

1: a = 0x7fff619813c0, b = 0x7fff619813b8, c = 0x7fff619813b0, i = 0x7fff619813ac
a is simply the starting point, b is then advanced down the stack 8 bytes since it is a point it will contain 8 bytes instead of 4, c also advances 8 bytes down from b since it is also a pointer then i only advances 4 bytes down because it is just an int and not an int pointer

---

printf("2: b = %p, c = %p\n", b, c);

2: b = 0x101e008b0, c = 0x7fff61982720
b is different from b above because it is not de-referenced and therefore printing the address will print the address on the heap and not the stack

---

printf("3: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n", a[0], a[1], a[2], a[3]);

3: a[0] = 200, a[1] = 101, a[2] = 102, a[3] = 103
a[0] = 200 because c is assigned to a and c[0] is directly assigned 200
a[1]-a[3] = 101 + i simply as assigned in the loop

---

printf("4: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n", a[0], a[1], a[2], a[3]);

4: a[0] will still be 200, a[1] will be 300 since a[1] is assigning it directly, a[2] is assigned 301 by adding 2 to the address of c and the dereferencing it and assigning it the value of 301, a[3] will be 302 because you are adding 3 to c which is a pointer so it will be assigned the value of 302 since it is deferenced

---

printf("5: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n", a[0], a[1], a[2], a[3]);

5: a[0] = 200, a[1] = 400 because you assign the c pointer to be c + 1 which moves it to element 1 in the array of a, a[2] = 301 and a[3] is 302 as they are both not modifed

---

printf("6: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n", a[0], a[1], a[2], a[3]);

6: a[0] = 200, still not modified, a[1] = ?, type casting the c pointer to char * will change the type of it and then add 1 to that char * and then type cast it back to an int * so the value is difficult to guess, a[2] = this is a value of memory leak since you are converting from char * to int * and then assigning the value of the pointer

---

printf("7: b = %p, c = %p\n", b, c);

7: b = 0x7fff616f1454, c = 0x7fff616f1451
b = 0x7fff616f1454 because you are taking a which is a pointer to 0x7fff616f1450 and adding 1 to it which will put you at 0x7fff616f1454 c is 0x7fff616f1451 because you are typecasting a as a char * and incrementing it by 1 which in char is just 1 byte so it will be 0x7fff616f1451 when casting it back as a int *