1-4 is completed by implementing functions in play.c.
5
For implementation in 3, the values in one should be the same as those in two (&ca == &pa, &(ca[0]) == &(pa[0]), etc.). It is because array names passed into the function declaration as parameters are treated as pointers. "Instead of passing a copy of the array, the compiler just passes its address." Therefore, the value of &(ca[0]), &(pa[0]), &ga and &(ga[0]) are all the same since they are actually the address pointing to the starting location of the global char array ga. char ca[] here is the same as char *pa.
Specifically, &ca and &pa is the address of the pointer pointing to the starting location of the array, so they are not equal to &ga as it is understood as the address of the pointer. &(ca[0]), on the other hand, is the actual address of the first element in the char array ga.
Moreover, pa++ is equal to &(ca[1]), &(pa[1]) and &(ga[1]). This is the magic of pointer arithmetic. The continuous address of (ca[0]) is (ca[1]), so pa++ is the next address of the location where pa is pointing.
The locally declared ga and pa would not affect the actual output since the scope is limited to the main function, and it will not make an alias with the formal parameter in function one and two.
The actual result printed out as the executable produces:

```
Value of &ca: 0x7fffa3998878
Value of &(ca[0]): 0x55766dec9010
Value of &(ca[1]):
0x55766dec9011
Value of &pa: 0x7fffa3998878
Value of &(pa[0]): 0x55766dec9010
Value of &(pa[1]): 0x55766dec9011
Value of ++pa:
0x55766dec9011
Value of &ga: 0x55766dec9010
Value of &(ga[0]): 0x55766dec9010
Value of &(ga[1]): 0x55766dec9011
```